Package ‘oce’

January 28, 2018

Type Package

Title Analysis of Oceanographic Data

Version 0.9-23

Maintainer Dan Kelley <Dan.Kelley@Dal.Ca>

Depends R (>= 2.15), utils, methods, gsw, testthat

Suggests ocedata, foreign, ncdf4, tiff, akima, RSQLite, DBI, rgdal (>= 1.1-9), R.utils, knitr, rmarkdown, marmap, lubridate

BugReports https://github.com/dankelley/oce/issues

Description Supports the analysis of Oceanographic data, including 'ADCP' measurements, measurements made with 'argo' floats, 'CTD' measurements, sectional data, sea-level time series, coastline and topographic data, etc. Provides specialized functions for calculating seawater properties such as potential temperature in either the 'UNESCO' or 'TEOS-10' equation of state. Produces graphical displays that conform to the conventions of the Oceanographic literature.

License GPL (>= 2)

Encoding UTF-8

URL https://dankelley.github.io/oce

LazyData false

RoxygenNote 6.0.1

BuildVignettes true

VignetteBuilder knitr

NeedsCompilation yes

Author Dan Kelley [aut, cre],
Clark Richards [aut],
Chantelle Layton [ctb] (curl() coauthor),
British Geological Survey [ctb, cph] (magnetic-field subroutine)

Repository CRAN

Date/Publication 2018-01-28 21:13:31 UTC

R topics documented:

abbreviateTimeLabels *Abbreviate a vector of times by removing commonalities*

Description

Abbreviate a vector of times by removing commonalities (e.g. year)

Usage

```
abbreviateTimeLabels(t, ...)
```

Arguments

`t`	vector of times.
`...`	optional arguments passed to the `format`, e.g. `format`.

Value

None.

Author(s)

Dan Kelley, with help from Clark Richards

See Also

This is used by various functions that draw time labels on axes, e.g. `plot,adp-method`.

addColumn *Add a Column to the Data Slot of an Oce object [deprecated]*

Description

WARNING: This function will be removed soon; see oce-deprecated. Use `oceSetData` instead of the present function.

Usage

```
addColumn(x, data, name)
```

Arguments

`x`	A `ctd` object, e.g. as read by `read.ctd`.
`data`	the data. The length of this item must match that of the existing data entries in the `data` slot).
`name`	the name of the column.

Value

An object of `class` oce, with a new column.

Author(s)

Dan Kelley

See Also

Please use `oceSetData` instead of the present function.

Other functions that will be removed soon: `ctdAddColumn`, `ctdUpdateHeader`, `mapMeridians`, `mapZones`, `oce.as.POSIXlt`

---

adp *ADP (acoustic-doppler profiler) dataset*

Description

This is degraded subsample of measurements that were made with an upward-pointing ADP manufactured by Teledyne-RDI, as part of the St Lawrence Internal Wave Experiment (SLEIWEX).

Usage

```
data(adp)
```

Source

This file came from the SLEIWEX-2008 experiment.

See Also

Other datasets provided with oce: `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

Examples

```
## Not run:
library(oce)
data(adp)

# Velocity components.  (Note: we should probably trim some bins at top.)
plot(adp)

# Note that tides have moved the mooring.
plot(adp, which=15:18)

## End(Not run)
```

---

adp-class *Class to Hold ADP (ADCP) Data*

Description

This class stores data from acoustic Doppler profilers. Some manufacturers call these ADCPs, while others call them ADPs; here the shorter form is used by analogy to ADVs.

Details

The `metadata` slot contains various items relating to the dataset, including source file name, sampling rate, velocity resolution, velocity maximum value, and so on. Some of these are particular to particular instrument types, and prudent researchers will take a moment to examine the whole contents of the metadata, either in summary form (with `str(adp[["metadata"]])`) or in detail (with `adp[["metadata"]]`). Perhaps the most useful general properties are `adp[["bin1Distance"]]` (the distance, in metres, from the sensor to the bottom of the first bin), `adp[["cellSize"]]` (the cell height, in metres, in the vertical direction, *not* along the beam), and `adp[["beamAngle"]]` (the angle, in degrees, between beams and an imaginary centre line that bisects all beam pairs).

The diagram provided below indicates the coordinate-axis and beam-numbering conventions for three- and four-beam ADP devices, viewed as though the reader were looking towards the beams being emitted from the transducers.

The bin geometry of a four-beam profiler is illustrated below, for `adp[["beamAngle"]]` equal to 20 degrees, `adp[["bin1Distance"]]` equal to 2m, and `adp[["cellSize"]]` equal to 1m. In the diagram, the viewer is in the plane containing two beams that are not shown, so the two visible beams are separated by 40 degrees. Circles indicate the centres of the range-gated bins within the beams. The lines enclosing those circles indicate the coverage of beams that spread plus and minus 2.5 degrees from their centreline.

Note that `adp[["oceCoordinate"]]` stores the present coordinate system of the object, and it has possible values `"beam"`, `"xyz"`, `"sfm"` or `"enu"`. (This should not be confused with `adp[["originalCoordinate"]]`, which stores the coordinate system used in the original data file.)

The `data` slot holds some standardized items, and many that vary from instrument to instrument. One standard item is `adp[["v"]]`, a three-dimensional numeric array of velocities in m/s. In this matrix, the first index indicates time, the second bin number, and the third beam number. The meaning of beams number depends on whether the object is in beam coordinates, frame coordinates, or earth coordinates. For example, if in earth coordinates, then beam 1 is the eastward component of velocity. Thus, for example,

```
library(oce)
data(adp)
t <- adp[['time']]
d <- adp[['distance']]
eastward <- adp[['v']][,,1]
imagep(t, d, eastward, missingColor="gray")
```

plots an image of the eastward component of velocity as a function of time (the x axis) and distance from sensor (y axis), since the adp dataset is in earth coordinates. Note the semidurnal tidal signal, and the pattern of missing data at the ocean surface (gray blotches at the top).

Corresponding to the velocity array are two arrays of type raw, and identical dimension, accessed by `adp[["a"]]` and `adp[["q"]]`, holding measures of signal strength and data quality quality, respectively. (The exact meanings of these depend on the particular type of instrument, and it is assumed that users will be familiar enough with instruments to know both the meanings and their practical consequences in terms of data-quality assessment, etc.)

In addition to the arrays, there are time-based vectors. The vector `adp[["time"]]` (of length equal to the first index of `adp[["v"]]`, etc.) holds times of observation. Depending on type of instrument and its configuration, there may also be corresponding vectors for sound speed (`adp[["soundSpeed"]]`), pressure (`adp[["pressure"]]`), temperature (`adp[["temperature"]]`), heading (`adp[["heading"]]`) pitch (`adp[["pitch"]]`), and roll (`adp[["roll"]]`), depending on the setup of the instrument.

The precise meanings of the data items depend on the instrument type. All instruments have `v` (for velocity), `q` (for a measure of data quality) and `a` (for a measure of backscatter amplitude, also called echo intensity). Teledyne-RDI profilers have an additional item g (for percent-good).

VmDas-equipped Teledyne-RDI profilers additional navigation data, with details listed in the table below; note that the RDI documentation [2] and the RDI gui use inconsistent names for most items.

Oce name	**RDI doc name**	**RDI GUI name**
`avgSpeed`	Avg Speed	Speed/Avg/Mag
`avgMagnitudeVelocityEast`	Avg Mag Vel East	?
`avgMagnitudeVelocityNorth`	Avg Mag Vel North	?
`avgTrackMagnetic`	Avg Track Magnetic	Speed/Avg/Dir (?)
`avgTrackTrue`	Avg Track True	Speed/Avg/Dir (?)
`avgTrueVelocityEast`	Avg True Vel East	?
`avgTrueVelocityNorth`	Avg True Vel North	?
`directionMadeGood`	Direction Made Good	Speed/Made Good/Dir
`firstLatitude`	First latitude	Start Lat
`firstLongitude`	First longitude	Start Lon
`firstTime`	UTC Time of last fix	End Time
`lastLatitude`	Last latitude	End Lat
`lastLongitude`	Last longitude	End Lon
`lastTime`	UTC Time of last fix	End Time
`numberOfHeadingSamplesAveraged`	Number heading samples averaged	?

`numberOfMagneticTrackSamplesAveraged`	Number of magnetic track samples averaged	?
`numberOfPitchRollSamplesAvg`	Number of magnetic track samples averaged	?
`numberOfSpeedSamplesAveraged`	Number of speed samples averaged	?
`numberOfTrueTrackSamplesAvg`	Number of true track samples averaged	?
`primaryFlags`	Primary Flags	?
`shipHeading`	Heading	?
`shipPitch`	Pitch	?
`shipRoll`	Roll	?
`speedMadeGood`	Speed Made Good	Speed/Made Good/Mag
`speedMadeGoodEast`	Speed MG East	?
`speedMadeGoodNorth`	Speed MG North	?

For Teledyne-RDI profilers, there are four three-dimensional arrays holding beamwise data. In these, the first index indicates time, the second bin number, and the third beam number (or coordinate number, for data in `xyz`, `sfm`, `enu` or `other` coordinate systems). In the list below, the quoted phrases are quantities as defined in Figure 9 of reference 1.

- `v` is "velocity" in m/s, inferred from two-byte signed integer values (multiplied by the scale factor that is stored in `velocityScale` in the metadata).
- `q` is "correlation magnitude" a one-byte quantity stored as type `raw` in the object. The values may range from 0 to 255.
- `a` is "backscatter amplitude", also known as "echo intensity" a one-byte quantity stored as type `raw` in the object. The values may range from 0 to 255.
- `g` is "percent good" a one-byte quantity stored as `raw` in the object. The values may range from 0 to 100.

Finally, there is a vector `adp[["distance"]]` that indicates the bin distances from the sensor, measured in metres along an imaginary centre line bisecting beam pairs. The length of this vector equals `dim(adp[["v"]])[2]`.

The `processingLog` slot is in standard form and needs little comment.

Teledyne-RDI Sentinel V ADCPs

As of 2016-09-27 there is provisional support for the TRDI "SentinelV" ADCPs, which are 5 beam ADCPs with a vertical centre beam. Relevant vertical beam fields are called `adp[["vv"]]`, `adp[["va"]]`, `adp[["vq"]]`, and `adp[["vg"]]` in analogy with the standard 4-beam fields.

Accessing and altering information within `adp-class` objects

Extracting values Matrix data may be accessed as illustrated above, e.g. or an adp object named `adv`, the data are provided by `adp[["v"]]`, `adp[["a"]]`, and `adp[["q"]]`. As a convenience, the last two of these can be accessed as numeric (as opposed to raw) values by e.g. `adp[["a", "numeric"]]`. The vectors are accessed in a similar way, e.g. `adp[["heading"]]`, etc. Quantities in the `metadata` slot are also available by name, e.g. `adp[["velocityResolution"]]`, etc.

Assigning values. This follows the standard form, e.g. to increase all velocity data by 1 cm/s, use `adp[["v"]] <- 0.01 + adp[["v"]]`.

Overview of contents The `show` method (e.g. `show(d)`) displays information about an ADP object named `d`.

Dealing with suspect data

There are many possibilities for confusion with `adp` devices, owing partly to the flexibility that manufacturers provide in the setup. Prudent users will undertake many tests before trusting the details of the data. Are mean currents in the expected direction, and of the expected magnitude, based on other observations or physical constraints? Is the phasing of currents as expected? If the signals are suspect, could an incorrect scale account for it? Could the transformation matrix be incorrect? Might the data have exceeded the maximum value, and then "wrapped around" to smaller values? Time spent on building confidence in data quality is seldom time wasted.

References

1. Teledyne-RDI, 2007. *WorkHorse commands and output data format.* P/N 957-6156-00 (November 2007).

2. Teledyne-RDI, 2012. *VmDas User's Guide, Ver. 1.46.5*.

See Also

A file containing ADP data is usually recognized by Oce, and so `read.oce` will usually read the data. If not, one may use the general ADP function `read.adp` or specialized variants `read.adp.rdi`, `read.adp.nortek` or `read.adp.sontek` or `read.adp.sontek.serial`.

ADP data may be plotted with `plot,adp-method`, which is a generic function so it may be called simply as `plot`.

Statistical summaries of ADP data are provided by the generic function `summary`, while briefer overviews are provided with `show`.

Conversion from beam to xyz coordinates may be done with `beamToXyzAdp`, and from xyz to enu (east north up) may be done with `xyzToEnuAdp`. `toEnuAdp` may be used to transfer either beam or xyz to enu. Enu may be converted to other coordinates (e.g. aligned with a coastline) with `enuToOtherAdp`.

Other classes provided by oce: `adv-class`, `argo-class`, `bremen-class`, `cm-class`, `coastline-class`, `ctd-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `oce-class`, `odf-class`, `rsk-class`, `sealevel-class`, `section-class`, `topo-class`, `windrose-class`

Other things related to `adp` data: `[[,adp-method`, `[[<-,adp-method`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

adpEnsembleAverage — *Ensemble Average an ADP Object in Time*

Description

Ensemble averaging of adp objects is often necessary to reduce the uncertainty in velocity estimates from single pings. Many types of ADPs can be configured to perform the ensemble averaging during the data collection, due to memory limitations for long deployments. In cases where the instrument is not memory limited, it may be desirable to perform the ensemble averaging during post-processing, thereby reducing the overall size of the data set and decreasing the uncertainty of the velocity estimates (by averaging out Doppler noise).

Usage

```
adpEnsembleAverage(x, n = 5, leftover = FALSE, na.rm = TRUE, ...)
```

Arguments

`x`	an adp object, i.e. one inheriting from `adp-class`.
`n`	number of pings to average together.
`leftover`	a logical value indicating how to proceed in cases where `n` does not divide evenly into the number of ensembles in `x`. If `leftover` is `FALSE` (the default) then any extra ensembles at the end of `x` are ignored. Otherwise, they are used to create a final ensemble in the returned value.
`na.rm`	a logical value indicating whether NA values should be stripped before the computation proceeds
`...`	extra arguments to be passed to the `mean()` function.

Value

A reduced object of `adp-class` with ensembles averaged as specified. E.g. for an adp object with 100 pings and `n=5` the number of rows of the data arrays will be reduced by a factor of 5.

Author(s)

Clark Richards and Dan Kelley

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

Examples

```
library(oce)
data(adp)
adpAvg <- adpEnsembleAverage(adp, n=2)
plot(adpAvg)
```

---

adv *ADV (acoustic-doppler velocimeter) dataset*

Description

This adv-class object is a sampling of measurements made with a Nortek Vector acoustic Doppler velocimeter deployed as part of the St Lawrence Internal Wave Experiment (SLEIWEX). Various identifying features have been redacted.

Usage

```
data(adv)
```

Source

This file came from the SLEIWEX-2008 experiment.

See Also

Other datasets provided with oce: adp, argo, cm, coastlineWorld, colors, ctdRaw, ctd, echosounder, landsat, lisst, lobo, met, rsk, sealevelTuktoyaktuk, sealevel, section, topoWorld, wind

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv-class, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

Examples

```
## Not run:
library(oce)
data(adv)

# Velocity time-series
plot(adv)

# Spectrum of upward component of velocity, with ``turbulent'' reference line
s <- spectrum(adv[["v"]][,3],plot=FALSE)
plot(log10(s$freq), log10(s$spec), type='l')
for (a in seq(-20, 20, by=1))
    abline(a=a, b=-5/3, col='gray', lty='dotted')
```

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

---

adv-class *Class to Hold adv Data*

Description

This class holds data from acoustic-Doppler velocimeters.

Details

The metadata slot contains various items relating to the dataset, including source file name, sampling rate, velocity resolution and scale, etc. The processingLog is in standard form and needs little comment. The data slot holds a numeric matrix v of velocities in m/s, with the first index indicating time and the second indicating beam number. The meanings of the beams depends on whether the object is in beam coordinates, frame coordinates, or earth coordinates. The data slot also contains identically-dimensioned raw matrices a and q, holding measures of signal strength and data quality quality, respectively. It also contains a series of vectors, e.g. time, temperature and pressure, etc., depending on what sensors are included in the package. For all of these quantities, the details can be different for different instrument types, and it is assumed that the user will be familiar with the details.

Data may be extracted with [[,adv-method and inserted with [[<-,adv-method. Type ?"[[,adv-method" or ?"[[<-,adv-method" to learn more.

See Also

A file containing ADV data is usually recognized by Oce, and so read.oce will usually read the data. If not, one may use the general ADV function read.adv or specialized variants read.adv.nortek, read.adv.sontek.adr or read.adv.sontek.text.

ADV data may be plotted with plot,adv-method function, which is a generic function so it may be called simply as plot(x), where x is an object inheriting from adv-class.

Statistical summaries of ADV data are provided by the generic function summary,adv-method.

Conversion from beam to xyz coordinates may be done with beamToXyzAdv, and from xyz to enu (east north up) may be done with xyzToEnuAdv. toEnuAdv may be used to transfer either beam or xyz to enu. Enu may be converted to other coordinates (e.g. aligned with a coastline) with enuToOtherAdv.

Other classes provided by oce: adp-class, argo-class, bremen-class, cm-class, coastline-class, ctd-class, echosounder-class, lisst-class, lobo-class, met-class, oce-class, odf-class, rsk-class, sealevel-class, section-class, topo-class, windrose-class

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

Examples

```
data(adv)
adv[["v"]] <- 0.001 + adv[["v"]] # add 1mm/s to all velocity components
```

---

airRho *Air density*

Description

Compute, ρ, the *in-situ* density of air.

Usage

```
airRho(temperature, pressure, humidity)
```

Arguments

`temperature`	*in-situ* temperature [°C]
`pressure`	pressure in Pa (NOT kPa) – ignored at present
`humidity`	ignored at present

Details

This will eventually be a proper equation of state, but for now it's just returns something from wikipedia (i.e. not trustworthy), and not using humidity.

Value

In-situ air density [kg/m^3].

Author(s)

Dan Kelley

References

National Oceanographic and Atmospheric Agency, 1976. U.S. Standard Atmosphere, 1976. NOAA-S/T 76-1562. (A PDF of this document may be available at `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/` or `http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA035728` although neither link has proven to be reliable.)

Examples

```
degC <- seq(0,30,length.out=100)
p <- seq(98,102,length.out=100) * 1e3
contour(x=degC, y=p, z=outer(degC,p,airRho), labcex=1)
```

amsr-class *Class to Hold amsr Data*

Description

The Advanced Microwave Scanning Radiometer (AMSR-2) is in current operation on the Japan Aerospace Exploration Agency (JAXA) GCOM-W1 space craft, launched in May 2012. Data are processed by Remote Sensing Systems. The satellite completes an ascending and descending pass during local daytime and nighttime hours respectively. Each daily file contains 7 daytime and 7 nighttime maps of variables named as follows within the `data` slot of amsr objects: `timeDay`, `SSTDay`, `LFwindDay` (wind at 10m sensed in the 10.7GHz band), `MFwindDay` (wind at 10m sensed at 18.7GHz), `vaporDay`, `cloudDay`, and `rainDay`, along with similarly-named items that end in `Night`. See [1] for additional information on the instrument, how to cite the data source in a paper, etc.

Details

The bands are stored in `raw` form, to save storage. The accessor function `[[,amsr-method` can provide these values in `raw` form or in physical units; `plot,amsr-method`, and `summary,amsr-method` work with physical units.

Author(s)

Dan Kelley and Chantelle Layton

References

1. Information on the satellite, how to cite the data, etc. is provided at `http://www.remss.com/missions/amsr/`.

2. A simple interface for viewing and downloading data is at `http://images.remss.com/amsr/amsr2_data_daily.html`.

See Also

`landsat-class` for handling data from the Landsat-8 satellite.

Other things related to `amsr` data: `[[<-,amsr-method`, `composite,amsr-method`, `download.amsr`, `plot,amsr-method`, `read.amsr`, `subset,amsr-method`, `summary,amsr-method` `angleRemap` *Convert angles from 0:360 to -180:180*

Description

This is mostly used for instrument heading angles, in cases where the instrument is aligned nearly northward, so that small variations in heading (e.g. due to mooring motion) can yield values that swing from small angles to large angles, because of the modulo-360 cut point. The method is to use the cosine and sine of the angle in order to find "x" and "y" values on a unit circle, and then to use `atan2` to infer the angles.

Usage

```
angleRemap(theta)
```

Arguments

`theta`	an angle (in degrees) that is in the range from 0 to 360 degrees

Value

A vector of angles, in the range -180 to 180.

Author(s)

Dan Kelley

Examples

```
library(oce)
## fake some heading data that lie near due-north (0 degrees)
n <- 20
heading <- 360 + rnorm(n, sd=10)
heading <- ifelse(heading > 360, heading - 360, heading)
x <- 1:n
plot(x, heading, ylim=c(-10, 360), type='l', col='lightgray', lwd=10)
lines(x, angleRemap(heading))
```

applyMagneticDeclination *Earth magnetic declination*

Description

Instruments that use magnetic compasses to determine current direction need to have corrections applied for magnetic declination, to get currents with the y component oriented to geographic, not magnetic, north. Sometimes, and for some instruments, the declination is specified when the instrument is set up, so that the velocities as recorded are already. Other times, the data need to be adjusted. This function is for the latter case.

Usage

```
applyMagneticDeclination(x, declination = 0, debug = getOption("oceDebug"))
```

Arguments

`x`	an oce object.
`declination`	magnetic declination (to be added to the heading)
`debug`	a debugging flag, set to a positive value to get debugging.

Value

Object, with velocity components adjusted to be aligned with geographic north and east.

Author(s)

Dan Kelley

References

'https://www.ngdc.noaa.gov/IAGA/vmod/igrf.html'

See Also

Use `magneticField` to determine the declination, inclination and intensity at a given spot on the world, at a given time.

Other things related to magnetism: `magneticField`

approx3d — *Trilinear interpolation in a 3D array*

Description

Interpolate within a 3D array, using the trilinear approximation.

Usage

```
approx3d(x, y, z, f, xout, yout, zout)
```

Arguments

`x`	vector of x values for grid (must be equi-spaced)
`y`	vector of y values for grid (must be equi-spaced)
`z`	vector of z values for grid (must be equi-spaced)
`f`	matrix of rank 3, with the gridd values mapping to the `x` values (first index of `f`), etc.
`xout`	vector of x values for output.
`yout`	vector of y values for output (length must match that of `xout`).
`zout`	vector of z values for output (length must match that of `xout`).

Details

Trilinear interpolation is used to interpolate within the `f` array, for those (`xout`, `yout` and `zout`) triplets that are inside the region specified by `x`, `y` and `z`. Triplets that lie outside the range of `x`, `y` or `z` result in `NA` values.

Value

A vector of interpolated values (or `NA` values), with length matching that of `xout`.

Author(s)

Dan Kelley and Clark Richards

Examples

```
## set up a grid
library(oce)
n <- 5
x <- seq(0, 1, length.out=n)
y <- seq(0, 1, length.out=n)
z <- seq(0, 1, length.out=n)
f <- array(1:n^3, dim=c(length(x), length(y), length(z)))
## interpolate along a diagonal line
m <- 100
```

```
xout <- seq(0, 1, length.out=m)
yout <- seq(0, 1, length.out=m)
zout <- seq(0, 1, length.out=m)
approx <- approx3d(x, y, z, f, xout, yout, zout)
## graph the results
plot(xout, approx, type='l')
points(xout[1], f[1, 1, 1])
points(xout[m], f[n,n,n])
```

---

argo *ARGO float dataset*

Description

This holds data from ARGO 6900388 in the North Atlantic.

Details

To quote Argo's website: "These data were collected and made freely available by the International Argo Program and the national programs that contribute to it. (http://www.argo.ucsd.edu, http://argo.jcommops.org). The Argo Program is part of the Global Ocean Observing System."

Below is the official citation (note that this DOI has web links for downloads): Argo (2017). Argo float data and metadata from Global Data Assembly Centre (Argo GDAC) - Snapshot of Argo GDAC of July, 8st 2017. SEANOE. http://doi.org/10.17882/42182#50865

Source

This file was downloaded using the unix command

```
ftp ftp://ftp.ifremer.fr/ifremer/argo/dac/bodc/6900388/6900388_prof.nc
```

issued on 2017 July 7.

See Also

Other datasets provided with oce: adp, adv, cm, coastlineWorld, colors, ctdRaw, ctd, echosounder, landsat, lisst, lobo, met, rsk, sealevelTuktoyaktuk, sealevel, section, topoWorld, wind

Other things related to argo data: [[,argo-method, [[<-,argo-method, argo-class, argoGrid, argoNames2oceNames, as.argo, handleFlags,argo-method, plot,argo-method, read.argo, subset,argo-method, summary,argo-method

Examples

```
## Not run:
library(oce)
data(argo)
summary(argo)
data(coastlineWorld)
plot(argo, which="trajectory", coastline=coastlineWorld)

## End(Not run)
```

`argo-class` *Class to hold Argo data*

Description

This class stores data from argo floats.

Details

An argo object may be read with `read.argo` or created with `as.argo`. Argo data can be gridded to constant pressures with `argoGrid`. Plots can be made with `plot,argo-method`, while `summary,argo-method` produces statistical summaries and show produces overviews. The usual oce generic functions are available, e.g. `[[,argo-method` may be used to extract data, and `[[<-,argo-method` may be used to insert data.

See `http://www.argo.ucsd.edu/Gridded_fields.html` for some argo-related datasets that may be useful in a wider context.

Author(s)

Dan Kelley and Clark Richards

See Also

Other classes provided by oce: `adp-class`, `adv-class`, `bremen-class`, `cm-class`, `coastline-class`, `ctd-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `oce-class`, `odf-class`, `rsk-class`, `sealevel-class`, `section-class`, `topo-class`, `windrose-class`

Other things related to argo data: `[[,argo-method`, `[[<-,argo-method`, `argoGrid`, `argoNames2oceNames`, `argo`, `as.argo`, `handleFlags,argo-method`, `plot,argo-method`, `read.argo`, `subset,argo-method`, `summary,argo-method` argoGrid *Grid Argo float data*

Description

Grid an Argo float, by interpolating to fixed pressure levels. The gridding is done with `approx`. If there is sufficient user demand, other methods may be added, by analogy to `sectionGrid`.

Usage

```
argoGrid(argo, p, debug = getOption("oceDebug"), ...)
```

Arguments

`argo`	A `argo` object to be gridded.
`p`	Optional indication of the pressure levels to which interpolation should be done. If this is not supplied, the pressure levels will be calculated based on the existing values, using medians. If `p="levitus"`, then pressures will be set to be those of the Levitus atlas, given by `standardDepths`, trimmed to the maximum pressure in `argo`. If `p` is a single numerical value, it is taken as the number of subdivisions to use in a call to `seq` that has range from 0 to the maximum pressure in `argo`. Finally, if a vector numerical values is provided, then it is used as is.
`debug`	A flag that turns on debugging. Higher values provide deeper debugging.
`...`	Optional arguments to `approx`, which is used to do the gridding.

Value

An object of `argo-class` that contains a pressure matrix with constant values along the first index.

A note about flags

Data-quality flags contained within the original object are ignored by this function, and the returned value contains no such flags. This is because such flags represent an assessment of the original data, not of quantities derived from those data. This function produces a warning to this effect. The recommended practice is to use `handleFlags` or some other means to deal with flags before calling the present function.

Author(s)

Dan Kelley and Clark Richards

See Also

Other things related to argo data: `[[,argo-method`, `[[<-,argo-method`, `argo-class`, `argoNames2oceNames`, `argo`, `as.argo`, `handleFlags,argo-method`, `plot,argo-method`, `read.argo`, `subset,argo-method`, `summary,argo-method`

Examples

```
library(oce)
data(argo)
g <- argoGrid(argo, p=seq(0, 100, 1))
par(mfrow=c(2,1))
t <- g[["time"]]
z <- -g[["pressure"]][,1]
## Set zlim because of spurious temperatures.
imagep(t, z, t(g[['temperature']]), ylim=c(-100,0), zlim=c(0,20))
imagep(t, z, t(g[['salinity']]), ylim=c(-100,0))
```

---

argoNames2oceNames   *Convert Argo Data Name to Oce Name*

Description

This function is used internally by `read.argo` to convert Argo-convention data names to oce-convention names. Users should not call this directly, since its return value may be changed at any moment (e.g. to include units as well as names).

Usage

```
argoNames2oceNames(names, ignore.case = TRUE)
```

Arguments

`names`	vector of character strings containing names in the Argo convention.
`ignore.case`	a logical value passed to `gsub`, indicating whether to ignore the case of input strings. The default is set to `TRUE` because some data files use lower-case names, despite the fact that the Argo documentation specifies upper-case.

Details

The inference of names was done by inspection of some data files, using [1] as a reference. It should be noted, however, that the data files examined contain some names that are not undocumented in [1], and others that are listed only in its changelog, with no actual definitions being given. For example, the files had six distinct variable names that seem to relate to phase in the oxygen sensor, but these are not translated by the present function because these variable names are not defined in [1], or not defined uniquely in [2].

The names are converted with `gsub`, using the `ignore.case` argument of the present function. The procedure is to first handle the items listed in the following table, with string searches anchored to the start of the string. After that, the qualifiers `_ADJUSTED`, `_ERROR` and `_QC`, are translated to `Adjusted`, `Error`, and `QC`, respectively.

Argo name	**oce name**
BBP	bbp

BETA_BACKSCATTERING	betaBackscattering
BPHASE_OXY	bphaseOxygen
CDOM	CDOM
CNDC	conductivity
CHLA	chlorophyllA
CP	beamAttenuation
CYCLE_NUMBER	cycleNumber
DATA_CENTRE	dataCentre
DATA_MODE	dataMode
DATA_STATE_INDICATOR	dataStateIndicator
DC_REFERENCE	DCReference
DIRECTION	direction
DOWN_IRRADIANCE	downwellingIrradiance
DOWNWELLING_PAR	downwellingPAR
FIRMWARE_VERSION	firmwareVersion
FIT_ERROR_NITRATE	fitErrorNitrate
FLUORESCENCE_CDOM	fluorescenceCDOM
FLUORESCENCE_CHLA	fluorescenceChlorophyllA
INST_REFERENCE	instReference
JULD	juld (and used to compute time)
JULD_QC_LOCATION	juldQCLocation
LATITUDE	latitude
LONGITUDE	longitude
MOLAR_DOXY	oxygenUncompensated
PH_IN_SITU_FREE	pHFree
PH_IN_SITU_TOTAL	pH
PI_NAME	PIName
PLATFORM_NUMBER	id
POSITION_ACCURACY	positionAccuracy
POSITIONING_SYSTEM	positioningSystem
PROFILE	profile
PROJECT_NAME	projectName
RAW_DOWNWELLING_IRRADIANCE	rawDownwellingIrradiance
RAW_DOWNWELLING_PAR	rawDownwellingPAR
RAW_UPWELLING_RADIANCE	rawUpwellingRadiance
STATION_PARAMETERS	stationParameters
TEMP	temperature
TEMP_CPU_CHLA	temperatureCPUChlorophyllA
TEMP_DOXY	temperatureOxygen
TEMP_NITRATE	temperatureNitrate
TEMP_PH	temperaturePH
TEMP_SPECTROPHOTOMETER_NITRATE	temperatureSpectrophotometerNitrate
TILT	tilt
TURBIDITY	turbidity
UP_RADIANCE	upwellingRadiance
UV_INTENSITY	UVIntensity
UV_INTENSITY_DARK_NITRATE	UVIntensityDarkNitrate
UV_INTENSITY_NITRATE	UVIntensityNitrate

VRS_PH	VRSpH
WMO_INST_TYPE	WMOInstType

Value

A character vector of the same length as names, but with replacements having been made for all known quantities.

References

1. Argo User's Manual Version 3.2, Dec 29th, 2015, available at https://archimer.ifremer.fr/doc/00187/29825/40575.pdf (but note that this is a draft; newer versions may have replaced this by now).

2. Argo list of parameters in an excel spreadsheet, available at https://www.argodatamgt.org/content/download/27444 (but note that the certificate at this website was noticed to be invalid on December 17, 2016, so exercise caution in downloading the file).

See Also

Other things related to argo data: [[,argo-method, [[<-,argo-method, argo-class, argoGrid, argo, as.argo, handleFlags,argo-method, plot,argo-method, read.argo, subset,argo-method, summary,argo-method

---

argShow *Show an argument to a function, e.g. for debugging*

Description

Show an argument to a function, e.g. for debugging

Usage

```
argShow(x, nshow = 2, last = FALSE, sep = "=")
```

Arguments

x	the argument
nshow	number of values to show at first (if length(x)> 1)
last	indicates whether this is the final argument to the function
sep	the separator between name and value

`as.adp` *Create an ADP Object*

Description

Create an ADP Object

Usage

```
as.adp(time, distance, v, a = NULL, q = NULL, orientation = "upward",
  coordinate = "enu")
```

Arguments

`time`	of observations in POSIXct format
`distance`	to centre of bins
`v`	array of velocities, with first index for time, second for bin number, and third for beam number
`a`	amplitude, a `raw` array with dimensions matching `u`
`q`	quality, a `raw` array with dimensions matching `u`
`orientation`	a string indicating sensor orientation, e.g. `"upward"` and `"downward"`
`coordinate`	a string indicating the coordinate system, `"enu"`, `"beam"`, `"xy"`, or `"other"`

Details

Construct an object of `adp-class`. Only a basic subset of the typical `data` slot is represented in the arguments to this function, on the assumption that typical usage in reading data is to set up a nearly-blank `adp-class` object, the `data` slot of which is then inserted. However, in some testing situations it can be useful to set up artificial adp objects, so the other arguments may be useful.

Value

An object of `adp-class`.

Author(s)

Dan Kelley

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

Examples

```
data(adp)
t <- adp[["time"]]
d <- adp[["distance"]]
v <- adp[["v"]]
a <- as.adp(time=t, distance=d, v=v)
## Not run:
plot(a)

## End(Not run)
```

---

as.argo *Coerce Data Into an Argo Dataset*

Description

Coerce a dataset into an argo dataset. This is not the right way to read official argo datasets, which are provided in NetCDF format and may be read with `read.argo`.

Usage

```
as.argo(time, longitude, latitude, salinity, temperature, pressure,
  units = NULL, id, filename = "", missingValue)
```

Arguments

`time`	vector of POSIXct times.
`longitude`	vector of longitudes.
`latitude`	vector of latitudes.
`salinity`	vector of salinities.
`temperature`	vector of temperatures.
`pressure`	vector of pressures.
`units`	optional list containing units. If `NULL`, the default, then `"degree east"` is used for `longitude`, `"degree north"` for `latitude`, `""` for `salinity`, `"ITS-90"` for `temperature`, and `"dbar"` for `pressure`.
`id`	identifier.
`filename`	source filename.
`missingValue`	Optional missing value, indicating data values that should be taken as `NA`.

Value

An object of `argo-class`.

Author(s)

Dan Kelley

See Also

The documentation for argo-class explains the structure of argo objects, and also outlines the other functions dealing with them.

Other things related to argo data: [[,argo-method, [[<-,argo-method, argo-class, argoGrid, argoNames2oceNames, argo, handleFlags,argo-method, plot,argo-method, read.argo, subset,argo-method, summary,argo-method

---

as.cm *Coerce data into a CM object*

Description

Coerce data into a CM object

Usage

```
as.cm(time, u = NULL, v = NULL, pressure = NULL, conductivity = NULL,
  temperature = NULL, salinity = NULL, longitude = NA, latitude = NA,
  filename = "", debug = getOption("oceDebug"))
```

Arguments

time	A vector of times of observation, or an oce object that holds time, in addition to either both u and v, or both directionTrue and speedHorizontal.
u	either a numerical vector containing the eastward component of velocity, in m/s, or an oce object that can can be coerced into a cm object. In the second case, the other arguments to the present function are ignored.
v	vector containing the northward component of velocity in m/s.
pressure	vector containing pressure in dbar. Ignored if the first argument contains an oce object holding pressure.
conductivity	Optional vector of conductivity. Ignored if the first argument contains an oce object holding pressure.
temperature	Optional vector of temperature. Ignored if the first argument contains an oce object holding temperature
salinity	Optional vector of salinity, assumed to be Practical Salinity. Ignored if the first argument contains an oce object holding salinity
longitude	Optional longitude in degrees East. Ignored if the first argument contains an oce object holding longitude.
latitude	Latitude in degrees North. Ignored if the first argument contains an oce object holding latitude.

filename	Optional source file name
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

See Also

Other things related to cm data: [[,cm-method, [[<-,cm-method, cm-class, cm, plot,cm-method, read.cm, subset,cm-method, summary,cm-method

---

as.coastline *Coerce Data into a Coastline Object*

Description

Coerces a sequence of longitudes and latitudes into a coastline dataset. This may be used when read.coastline cannot read a file, or when the data have been manipulated.

Usage

```
as.coastline(longitude, latitude, fillable = FALSE)
```

Arguments

longitude	the longitude in decimal degrees, positive east of Greenwich, or a data frame with columns named latitude and longitude, in which case these values are extracted from the data frame and the second argument is ignored.
latitude	the latitude in decimal degrees, positive north of the Equator.
fillable	boolean indicating whether the coastline can be drawn as a filled polygon.

Value

An object of class "coastline" (for details, see read.coastline).

Author(s)

Dan Kelley

See Also

Other things related to coastline data: [[,coastline-method, [[<-,coastline-method, coastline-class, coastlineBest, coastlineCut, coastlineWorld, download.coastline, plot,coastline-method, read.coastline.openstreetmap, read.coastline.shapefile, subset,coastline-method, summary,coastline-meth as.ctd *Coerce data into CTD object*

Description

Assemble data into a ctd-class dataset.

Usage

```
as.ctd(salinity, temperature = NULL, pressure = NULL, conductivity = NULL,
  scan = NULL, time = NULL, other = NULL, units = NULL, flags = NULL,
  missingValue = NULL, type = "", serialNumber = "", ship = NULL,
  cruise = NULL, station = NULL, startTime = NULL, longitude = NULL,
  latitude = NULL, deploymentType = "unknown", pressureAtmospheric = 0,
  sampleInterval = NA, profile = NULL, debug = getOption("oceDebug"))
```

Arguments

`salinity`	There are several distinct choices for `salinity`. (1) It can be a vector indicating the practical salinity through the water column. In that case, `as.ctd` employs the other arguments listed below. (2) it can be something (a data frame, a list or an `oce` object) from which practical salinity, temperature, pressure, and conductivity can be inferred. In this case, the relevant information is extracted and the other arguments to `as.ctd` are ignored, except for `pressureAtmospheric`. If the first argument has salinity, etc., in matrix form (as can happen with some objects of argo-class), then only the first column is used, and a warning to that effect is given, unless the `profile` argument is specified and then that specific profile is extracted. (3) It can be an object of rsk-class, (see "Converting rsk objects" for details). (4) It can be unspecified, in which case `conductivity` becomes a mandatory argument, because it will be needed for computing actual salinity, using swSCTp.
`temperature`	*in-situ* temperature [$^\circ deg$C], defined on the ITS-90 scale; see "Temperature units" in the documentation for swRho.
`pressure`	Vector of pressure values, one for each `salinity` and `temperature` pair, or just a single pressure, which is repeated to match the length of `salinity`.
`conductivity`	electrical conductivity ratio through the water column (optional). To convert from raw conductivity in milliSeimens per centimeter divide by 42.914 to get conductivity ratio (see Culkin and Smith, 1980).
`scan`	optional scan number. If not provided, this will be set to `1:length(salinity)`.
`time`	optional vector of times of observation
`other`	optional list of other data columns that are not in the standard list
`units`	an optional list containing units. If not supplied, defaults are set for `pressure`, `temperature`, `salinity`, and `conductivity`. Since these are simply guesses, users are advised strongly to supply `units`. See "Examples".

flags	if supplied, this is a list containing data-quality flags. The elements of this list must have names that match the data provided to the object.
missingValue	optional missing value, indicating data that should be taken as NA. Set to NULL to turn off this feature.
type	optional type of CTD, e.g. "SBE"
serialNumber	optional serial number of instrument
ship	optional string containing the ship from which the observations were made.
cruise	optional string containing a cruise identifier.
station	optional string containing a station identifier.
startTime	optional indication of the start time for the profile, which is used in some several plotting functions. This is best given as a POSIXt time, but it may also be a character string that can be converted to a time with as.POSIXct, using UTC as the timezone.
longitude	optional numerical value containing longitude in decimal degrees, positive in the eastern hemisphere. If this is a single number, then it is stored in the metadata slot of the returned value; if it is a vector of numbers, they are stored in data and a mean value is stored in metadata.
latitude	optional numerical value containing the latitude in decimal degrees, positive in the northern hemisphere. See the note on length, for the longitude argument.
deploymentType	character string indicating the type of deployment. Use "unknown" if this is not known, "profile" for a profile (in which the data were acquired during a downcast, while the device was lowered into the water column, perhaps also including an upcast; "moored" if the device is installed on a fixed mooring, "thermosalinograph" (or "tsg") if the device is mounted on a moving vessel, to record near-surface properties, or "towyo" if the device is repeatedly lowered and raised.
pressureAtmospheric	A numerical value (a constant or a vector), that is subtracted from pressure before storing it in the return value. (This altered pressure is also used in calculating salinity, if that is to be computed from conductivity, etc., using swSCTp (see salinity above).
sampleInterval	optional numerical value indicating the time between samples in the profile.
profile	optional positive integer specifying the number of the profile to extract from an object that has data in matrices, such as for some argo objects. Currently the profile argument is only utilized for argo-class objects.
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Value

An object of ctd-class.

Converting rsk objects

If the salinity argument is an object of rsk-class, then as.ctd passes it, pressureAtmospheric, longitude, latitude ship, cruise, station and deploymentType to rsk2ctd, which builds the ctd object that is returned by as.ctd. The other arguments to as.ctd are ignored in this instance, because rsk objects already contain their information. If required, any data or metadata element can be added to the value returned by as.ctd using oceSetData or oceSetMetadata, respectively.

The returned rsk-class object contains pressure in a form that may need to be adjusted, because rsk objects may contain either absolute pressure or sea pressure. This adjustment is handled automatically by as.ctd, by examination of the metadata item named pressureType (described in the documentation for read.rsk). Once the sea pressure is determined, adjustments may be made with the pressureAtmospheric argument, although in that case it is better considered a pressure adjustment than the atmospheric pressure.

rsk-class objects may store sea pressure or absolute pressure (the sum of sea pressure and atmospheric pressure), depending on how the object was created with as.rsk or read.rsk. However, ctd-class objects store sea pressure, which is needed for plotting, calculating density, etc. This poses no difficulties, however, because as.ctd automatically converts absolute pressure to sea pressure, if the metadata in the rsk-class object indicates that this is appropriate. Further alteration of the pressure can be accomplished with the pressureAtmospheric argument, as noted above.

Author(s)

Dan Kelley

References

Culkin, F., and Norman D. Smith, 1980. Determination of the concentration of potassium chloride solution having the same electrical conductivity, at 15 C and infinite frequency, as standard seawater of salinity 35.0000 ppt (Chlorinity 19.37394 ppt). *IEEE Journal of Oceanic Engineering*, **5**, pp 22-23.

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
library(oce)
## 1. fake data, with default units
pressure <- 1:50
temperature <- 10 - tanh((pressure - 20) / 5) + 0.02*rnorm(50)
salinity <- 34 + 0.5*tanh((pressure - 20) / 5) + 0.01*rnorm(50)
ctd <- as.ctd(salinity, temperature, pressure)
# Add a new column
fluo <- 5 * exp(-pressure / 20)
ctd <- oceSetData(ctd, name="fluorescence", value=fluo,
```

```
                  unit=list(unit=expression(mg/m^3), scale=""))
summary(ctd)

## 2. fake data, with supplied units (which are the defaults, actually)
ctd <- as.ctd(salinity, temperature, pressure,
    units=list(salinity=list(unit=expression(), scale="PSS-78"),
    temperature=list(unit=expression(degree*C), scale="ITS-90"),
    pressure=list(unit=expression(dbar), scale="")))
```

`as.echosounder` *Coerce Data into an Echosounder Object*

Description

Coerces a dataset into a echosounder dataset.

Usage

```
as.echosounder(time, depth, a, src = "", sourceLevel = 220,
  receiverSensitivity = -55.4, transmitPower = 0, pulseDuration = 400,
  beamwidthX = 6.5, beamwidthY = 6.5, frequency = 41800, correction = 0)
```

Arguments

`time`	times of pings
`depth`	depths of samples within pings
`a`	matrix of amplitudes
`src`	optional string indicating data source
`sourceLevel`	source level, in dB (uPa at 1m), denoted `sl` in [1 p15], where it is in units 0.1dB (uPa at 1m)
`receiverSensitivity`	receiver sensivity of the main element, in dB(counts/uPa), denoted `rs` in [1 p15], where it is in units of 0.1dB(counts/uPa)
`transmitPower`	transmit power reduction factor, in dB, denoted `tpow` in [1 p10], where it is in units 0.1 dB.
`pulseDuration`	duration of transmitted pulse in us
`beamwidthX`	x-axis -3dB one-way beamwidth in deg, denoted `bwx` in [1 p16], where the unit is 0.2 deg
`beamwidthY`	y-axis -3dB one-way beamwidth in deg, denoted `bwx` in [1 p16], where the unit is 0.2 deg
`frequency`	transducer frequency in Hz, denoted `fq` in [1 p16]
`correction`	user-defined calibration correction in dB, denoted `corr` in [1 p14], where the unit is 0.01dB.

Details

Creates an echosounder file. The defaults for e.g. `transmitPower` are taken from the `echosounder` dataset, and they are unlikely to make sense generally.

Value

An object of `class` `"echosounder"`; for details of this data type, see `echosounder-class`).

Author(s)

Dan Kelley

See Also

Other things related to `echosounder` data: `[[,echosounder-method`, `[[<-,echosounder-method`, `echosounder-class`, `echosounder`, `findBottom`, `plot,echosounder-method`, `read.echosounder`, `subset,echosounder-method`, `summary,echosounder-method`

`as.gps` *Coerce data into a GPS dataset*

Description

Coerces a sequence of longitudes and latitudes into a GPS dataset. This may be used when `read.gps` cannot read a file, or when the data have been manipulated.

Usage

```
as.gps(longitude, latitude, filename = "")
```

Arguments

`longitude`	the longitude in decimal degrees, positive east of Greenwich, or a data frame with columns named `latitude` and `longitude`, in which case these values are extracted from the data frame and the second argument is ignored.
`latitude`	the latitude in decimal degrees, positive north of the Equator.
`filename`	name of file containing data (if applicable).

Value

An object of `gps-class`.

Author(s)

Dan Kelley

See Also

Other things related to gps data: [[,gps-method, [[<-,gps-method, gps-class, plot,gps-method, read.gps, summary,gps-method

---

as.ladp *Coerce data into an ladp object*

Description

This function assembles vectors of pressure and velocity, possibly also with shears, salinity, temperature, etc.

Usage

```
as.ladp(longitude, latitude, station, time, pressure, u, v, uz, vz, salinity,
  temperature, ...)
```

Arguments

longitude	longitude in degrees east, or an oce object that contains the data otherwise given by longitude and the other arguments.
latitude	latitude in degrees east (use negative in southern hemisphere).
station	number or string indicating station ID.
time	time at the start of the profile, constructed by e.g. as.POSIXct.
pressure	pressure in decibars, through the water column.
u	eastward velocity (m/s).
v	northward velocity (m/s).
uz	vertical derivative of eastward velocity (1/s).
vz	vertical derivative of northward velocity (1/s).
salinity	salinity through the water column, in practical salinity units.
temperature	temperature through the water column.
...	optional additional data columns.

Value

An object of ladp-class.

Author(s)

Dan Kelley

See Also

Other things related to ladp data: [[,ladp-method, ladp-class, plot,ladp-method, summary,ladp-method `as.lisst` *Coerce Data Into a LISST Object*

Description

Coerce data into a lisst object If `data` contains fewer than 42 columns, an error is reported. If it contains more than 42 columns, only the first 42 are used. This is used by `read.lisst`, the documentation on which explains the meanings of the columns.

Usage

```
as.lisst(data, filename = "", year = 0, tz = "UTC", longitude = NA,
  latitude = NA)
```

Arguments

`data`	A table (or matrix) containing 42 columns, as in a LISST data file.
`filename`	Name of file containing the data.
`year`	Year in which the first observation was made. This is necessary because LISST timestamps do not indicate the year of observation. The default value is odd enough to remind users to include this argument.
`tz`	Timezone of observations. This is necessary because LISST timestamps do not indicate the timezone.
`longitude`	Longitude of observation.
`latitude`	Latitude of observation.

Value

An object of `lisst-class`.

Author(s)

Dan Kelley

See Also

Other things related to `lisst` data: `[[,lisst-method`, `[[<-,lisst-method`, `lisst-class`, `plot,lisst-method`, `read.lisst`, `summary,lisst-method` `as.lobo` *Coerce Data into a Lobo Object*

Description

Coerce a dataset into a lobo dataset.

Usage

```
as.lobo(time, u, v, salinity, temperature, pressure, nitrate, fluorescence,
  filename = "")
```

Arguments

`time`	vector of times of observation
`u`	vector of x velocity component observations
`v`	vector of y velocity component observations
`salinity`	vector of salinity observations
`temperature`	vector of temperature observations
`pressure`	vector of pressure observations
`nitrate`	vector of nitrate observationss
`fluorescence`	vector of fluoresence observations
`filename`	source filename

Value

An object of `lobo-class`.

Author(s)

Dan Kelley

See Also

Other things related to lobo data: `[[,lobo-method`, `[[<-,lobo-method`, `lobo-class`, `lobo`, `plot,lobo-method`, `subset,lobo-method`, `summary,lobo-method`

as.met *Coerce Data into met Object*

Description

Coerces a dataset into a met dataset. This fills in only a few of the typical data fields, so the returned object is much sparser than the output from `read.met`. Also, almost no metadata fields are filled in, so the resultant object does not store station location, units of the data, data-quality flags, etc. Anyone working with data from Environment Canada [2] is advised to use `read.met` instead of the present function.

Usage

```
as.met(time, temperature, pressure, u, v,
  filename = "(constructed from data)")
```

Arguments

`time`	Either a vector of observation times (or character strings that can be coerced into times) or the output from `canadaHCD::hcd_hourly` (see [1]).
`temperature`	vector of temperatures.
`pressure`	vector of pressures.
`u`	vector of eastward wind speed in m/s.
`v`	vector of northward wind speed in m/s.
`filename`	optional string indicating data source

Value

An object of `met-class`.

Author(s)

Dan Kelley

References

1. The `canadaHCD` package is in development by Gavin Simpson; see https://github.com/gavinsimpson/canadaHCD for instructions on how to download and install from GitHub.

2. Environment Canada website for Historical Climate Data http://climate.weather.gc.ca/index_e.html

See Also

Other things related to `met` data: `[[,met-method`, `[[<-,met-method`, `download.met`, `met-class`, `met`, `plot,met-method`, `read.met`, `subset,met-method`, `summary,met-method` as.oce *Coerce Something Into an Oce Object*

Description

Coerce Something Into an Oce Object

Usage

```
as.oce(x, ...)
```

Arguments

x	an item containing data. This may be data frame, list, or an oce object.
...	optional extra arguments, passed to conversion functions `as.coastline` or `ODF2oce`, if these are used.

Details

This function is limited and not intended for common use. In most circumstances, users should employ a function such as `as.ctd` to construct specialized oce sub-classes.

`as.oce` creates an oce object from data contained within its first argument, which may be a list, a data frame, or an object of `oce-class`. (In the last case, `x` is simply returned, without modification.)

If `x` is a list containing items named `longitude` and `latitude`, then `as.coastline` is called (with the specified ... value) to create a coastline object.

If `x` is a list created by `read_odf` from the (as yet unreleased) ODF package developed by the Bedford Institute of Oceanography, then `ODF2oce` is called (with no arguments other than the first) to calculate a return value. If the sub-class inference made by `ODF2oce` is incorrect, users should call that function directly, specifying a value for its `coerce` argument.

If `x` has not been created by `read_odf`, then the names of the items it contains are examined, and used to try to infer the proper return value. There are only a few cases (although more may be added if there is sufficient user demand). The cases are as follows.

- If `x` contains items named `temperature`, `pressure` and either `salinity` or `conductivity`, then an object of type `ctd-class` will be returned.
- If `x` contains columns named `longitude` and `latitude`, but no other columns, then an object of class `coastline-class` is returned.

Value

`as.oce` returns an object inheriting from `oce-class`.

`as.rsk` *Coerce Data Into a Rsk Object*

Description

Create a rsk object.

Usage

```
as.rsk(time, columns, filename = "", instrumentType = "rbr",
  serialNumber = "", model = "", sampleInterval = NA,
  debug = getOption("oceDebug"))
```

Arguments

`time`	a vector of times for the data.
`columns`	a list or data frame containing the measurements at the indicated times; see "Details".
`filename`	optional name of file containing the data.
`instrumentType`	type of instrument.
`serialNumber`	serial number for instrument.
`model`	instrument model type, e.g. `"RBRduo"`.
`sampleInterval`	sampling interval. If given as `NA`, then this is estimated as the median difference in times.
`debug`	a flag that can be set to `TRUE` to turn on debugging.

Details

The contents of `columns` are be copied into the `data` slot of the returned object directly, so it is critical that the names and units correspond to those expected by other code dealing with `rsk-class` objects. If there is a conductivity, it must be called `conductivity`, and it must be in units of mS/cm. If there is a temperature, it must be called `temperature`, and it must be an in-situ value recorded in ITS-90 units. And if there is a pressure, it must be *absolute* pressure (sea pressure plus atmospheric pressure) and it must be named `pressure`. No checks are made within `as.rsk` on any of these rules, but if they are broken, you may expect problems with any further processing.

Value

An object of `rsk-class` `"rsk"`.

Author(s)

Dan Kelley

See Also

Other things related to rsk data: [[,rsk-method, [[<-,rsk-method, plot,rsk-method, read.rsk, rsk-class, rskPatm, rskToc, rsk, subset,rsk-method, summary,rsk-method

---

as.sealevel *Coerce Data Into a Sealevel Object*

Description

Coerces a dataset (minimally, a sequence of times and heights) into a sealevel dataset. The arguments are based on the standard data format, as were described in a file formerly available at [1].

Usage

```
as.sealevel(elevation, time, header = NULL, stationNumber = NA,
  stationVersion = NA, stationName = NULL, region = NULL, year = NA,
  longitude = NA, latitude = NA, GMTOffset = NA, decimationMethod = NA,
  referenceOffset = NA, referenceCode = NA, deltat)
```

Arguments

elevation	a list of sea-level heights in metres, in an hourly sequence.
time	optional list of times, in POSIXct format. If missing, the list will be constructed assuming hourly samples, starting at 0000-01-01 00:00:00.
header	a character string as read from first line of a standard data file.
stationNumber	three-character string giving station number.
stationVersion	single character for version of station.
stationName	the name of station (at most 18 characters).
region	the name of the region or country of station (at most 19 characters).
year	the year of observation.
longitude	the longitude in decimal degrees, positive east of Greenwich.
latitude	the latitude in decimal degrees, positive north of the equator.
GMTOffset	offset from GMT, in hours.
decimationMethod	a coded value, with 1 meaning filtered, 2 meaning a simple average of all samples, 3 meaning spot readings, and 4 meaning some other method.
referenceOffset	?
referenceCode	?
deltat	optional interval between samples, in hours (as for the ts timeseries function). If this is not provided, and t can be understood as a time, then the difference between the first two times is used. If this is not provided, and t cannot be understood as a time, then 1 hour is assumed.

Value

An object of class "sealevel" (for details, see read.sealevel).

Author(s)

Dan Kelley

References

http://ilikai.soest.hawaii.edu/rqds/hourly.fmt (this link worked for years but failed at least temporarily on December 4, 2016).

See Also

The documentation for sealevel-class explains the structure of sealevel objects, and also outlines the other functions dealing with them.

Other things related to sealevel data: [[,sealevel-method, [[<-,sealevel-method, plot,sealevel-method, read.sealevel, sealevel-class, sealevelTuktoyaktuk, sealevel, subset,sealevel-method, summary,sealevel-method

Examples

```
library(oce)

# Construct a year of M2 tide, starting at the default time
# 0000-01-01T00:00:00.
h <- seq(0, 24*365)
elevation <- 2.0 * sin(2*pi*h/12.4172)
sl <- as.sealevel(elevation)
summary(sl)

# As above, but start at the Y2K time.
time <- as.POSIXct("2000-01-01") + h * 3600
sl <- as.sealevel(elevation, time)
summary(sl)
```

---

as.section *Create a Section*

Description

Create a section based on columnar data, or a set of oce-class objects that can be coerced to a section. There are three cases.

Case 1. If the first argument is a numerical vector, then it is taken to be the salinity, and factor is applied to station to break the data up into chunks that are assembled into ctd-class objects with as.ctd and combined to make a section-class object to be returned. This mode of operation is provided as a convenience for datasets that are already partly processed; if original CTD data are available, the next mode is preferred, because it permits the storage of much more data and metadata in the CTD object.

Case 2. If the first argument is a list containing oce objects, then those objects are taken as profiles of something. A requirement for this to work is that every element of the list contains both `longitude` and `latitude` in either the `metadata` or `data` slot (in the latter case, the mean value is recorded in the section object) and that every element also contains `pressure` in its `data` slot.

Case 3. If the first argument is a `argo-class` object, then the profiles it contains are turned into `ctd-class` objects, and these are assembled into a section to be returned.

Usage

```
as.section(salinity, temperature, pressure, longitude, latitude, station,
  sectionId = "")
```

Arguments

`salinity`	This may be a numerical vector, in which case it is interpreted as the salinity, and the other arguments are used for the other components of `ctd-class` objects. Alternatively, it may be one of a variety of other objects from which the CTD objects can be inferred, in which case the other arguments are ignored; see 'Details'.
`temperature`	Temperature, in a vector holding values for all stations.
`pressure`	Pressure, in a vector holding values for all stations.
`longitude`	Longitude, in a vector holding values for all stations.
`latitude`	Latitude, in a vector holding values for all stations.
`station`	Station identifiers, in a vector holding values for all stations.
`sectionId`	Section identifier.

Value

An object of `section-class`.

Author(s)

Dan Kelley

See Also

Other things related to section data: `[[,section-method`, `[[<-,section-method`, `handleFlags,section-method`, `plot,section-method`, `read.section`, `section-class`, `sectionAddStation`, `sectionGrid`, `sectionSmooth`, `sectionSort`, `section`, `subset,section-method`, `summary,section-method`

Examples

```
library(oce)
data(ctd)
## vector of names of CTD objects
fake <- ctd
fake[["temperature"]] <- ctd[["temperature"]] + 0.5
fake[["salinity"]] <- ctd[["salinity"]] + 0.1
fake[["longitude"]] <- ctd[["longitude"]] + 0.01
fake[["station"]] <- "fake"
sec1 <- as.section(c("ctd", "fake"))
summary(sec1)
## vector of CTD objects
ctds <- vector("list", 2)
ctds[[1]] <- ctd
ctds[[2]] <- fake
sec2 <- as.section(ctds)
summary(sec2)
## argo data (a subset)
data(argo)
sec3 <- as.section(subset(argo, profile<5))
summary(sec3)
```

---

as.topo — *Coerce Data into Topo Object*

Description

Coerce Data into Topo Object

Usage

```
as.topo(longitude, latitude, z, filename = "")
```

Arguments

`longitude`	Either a vector of longitudes (in degrees east, and bounded by -180 and 180), or a `bathy` object created by `getNOAA.bathy()` from the `marmap` package; in the second case, all other arguments are ignored.
`latitude`	A vector of latitudes.
`z`	A matrix of heights (positive over land).
`filename`	Name of data (used when called by `read.topo`.

Value

An object of `topo-class`.

Author(s)

Dan Kelley

See Also

Other things related to topo data: [[,topo-method, [[<-,topo-method, download.topo, plot,topo-method, read.topo, subset,topo-method, summary,topo-method, topo-class, topoInterpolate, topoWorld

---

as.unit *Convert a String to a Unit*

Description

Convert a String to a Unit

Usage

```
as.unit(u, default = list(unit = expression(), scale = ""))
```

Arguments

u	A character string indicating a variable name. The following names are recognized: "DBAR", "IPTS-68", "ITS-90", "PSS-78", and "UMOL/KG". All other names yield a return value equal to the value of the default argument.
default	A default to be used for the return value, if u is not a recognized string.

Details

This function is not presently used by any oce functions, and is provided as a convenience function for users.

Value

A list with elements unit, an expression, and scale, a string.

`as.windrose` *Create a Windrose Object*

Description

Create a wind-rose object, typically for plotting with plot,windrose-method.

Usage

```
as.windrose(x, y, dtheta = 15, debug = getOption("oceDebug"))
```

Arguments

x	The x component of wind speed (or stress) *or* an object of class met (see met-class), in which case the u and v components of that object are used for the components of wind speed, and y here is ignored.
y	The y component of wind speed (or stress).
dtheta	The angle increment (in degrees) within which to classify the data.
debug	A flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Value

An object of windrose-class that contains the standard oce slots named data, metadata and proxessingLog. The data slot contains

n the number of x values

x.mean the mean of the x values

y.mean the mean of the y values

theta the central angle (in degrees) for the class

count the number of observations in this class

mean the mean of the observations in this class

fivenum the fivenum vector for observations in this class (the min, the lower hinge, the median, the upper hinge, and the max)

Author(s)

Dan Kelley, with considerable help from Alex Deckmyn.

See Also

Other things related to windrose data: [[,windrose-method, [[<-,windrose-method, plot,windrose-method, summary,windrose-method, windrose-class

Examples

```
library(oce)
xcomp <- rnorm(360) + 1
ycomp <- rnorm(360)
wr <- as.windrose(xcomp, ycomp)
summary(wr)
plot(wr)
```

---

bcdToInteger *Decode BCD to integer*

Description

Decode BCD to integer

Usage

```
bcdToInteger(x, endian = c("little", "big"))
```

Arguments

`x`	a raw value, or vector of raw values, coded in binary-coded decimal.
`endian`	character string indicating the endian-ness ("big" or "little"). The PC/intel convention is to use "little", and so most data files are in that format.

Value

An integer, or list of integers.

Author(s)

Dan Kelley

Examples

```
library(oce)
twenty.five <- bcdToInteger(as.raw(0x25))
thirty.seven <- as.integer(as.raw(0x25))
```

beamName *Get names of Acoustic-Doppler Beams*

Description

Get names of Acoustic-Doppler Beams

Usage

```
beamName(x, which)
```

Arguments

x	An adp object, i.e. one inheriting from adp-class.
which	an integer indicating beam number.

Value

A character string containing a reasonable name for the beam, of the form "beam 1", etc., for beam coordinates, "east", etc. for enu coordinates, "u", etc. for "xyz", or "u'", etc., for "other" coordinates. The coordinate system is determined with x[["coordinate"]].

Author(s)

Dan Kelley

See Also

This is used by read.oce.

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv-class, adv, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu `beamToXyz` *Change ADV or ADP coordinate systems*

Description

Convert velocity data from an acoustic-Doppler velocimeter or acoustic-Doppler profiler from one coordinate system to another.

Usage

```
beamToXyz(x, ...)
```

Arguments

`x`	an adp or adv object, i.e. one inheriting from `adp-class` or `adv-class`.
`...`	extra arguments that are passed on to `beamToXyzAdp` or `beamToXyzAdv`.

Value

An object of the same type as `x`, but with velocities in xyz coordinates instead of beam coordinates.

Author(s)

Dan Kelley

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `enuToOtherAdv`, `enuToOther`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnuAdv`, `xyzToEnu` beamToXyzAdp *Convert ADP From Beam to XYZ Coordinates*

Description

Convert ADP velocity components from a beam-based coordinate system to a xyz-based coordinate system.

Usage

```
beamToXyzAdp(x, debug = getOption("oceDebug"))
```

Arguments

`x`	an object of class `"adp"`.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

The action depends on the type of object.

For a 3-beam `aquadopp` object, the beams are transformed into velocities using the matrix stored in the header.

For 4-beam `rdi` object, the beams are converted to velocity components using formulae from section 5.5 of *RD Instruments* (1998), viz. the along-beam velocity components B_1, B_2, B_3, and B_4 are used to calculate velocity components in a cartesian system referenced to the instrument using the following formulae: $u = ca(B_1 - B_2)$, $v = ca(B_4 - B_3)$, $w = -b(B_1 + B_2 + B_3 + B_4)$, and an estimate of the error in velocity is calculated using $e = d(B_1 + B_2 - B_3 - B_4)$

(Note that the multiplier on e is subject to discussion; RDI suggests one multiplier, but some oceanographers favour another.)

In the above, $c = 1$ if the beam geometry is convex, and $c = -1$ if the beam geometry is concave, $a = 1/(2\sin\theta)$, $b = 1/(4\cos\theta)$ and $d = a/\sqrt{2}$, where θ is the angle the beams make to the instrument “vertical”.

Value

An object with the first 3 velocity indices having been altered to represent velocity components in xyz (or instrument) coordinates. (For `rdi` data, the values at the 4th velocity index are changed to represent the "error" velocity.)

To indicate the change, the value of `metadata$oce.orientation` is changed from `beam` to `xyz`.

Author(s)

Dan Kelley

References

1. R D Instruments, 1998. *ADP Coordinate Transformation, formulas and calculations.* P/N 951-6079-00 (July 1998).

2. WHOI/USGS-provided Matlab code for beam-enu transformation `http://woodshole.er.usgs.gov/pubs/of2005-142

See Also

See `read.adp` for other functions that relate to objects of class `"adp"`.

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

`beamToXyzAdv` *Convert ADV from Beam to XYZ Coordinates*

Description

Convert ADV velocity components from a beam-based coordinate system to a xyz-based coordinate system.

Usage

```
beamToXyzAdv(x, debug = getOption("oceDebug"))
```

Arguments

`x`	an object of class `"adv"`.
`debug`	a flag that, if non-zero, turns on debugging. Higher values yield more extensive debugging.

Details

The coordinate transformation is done using the transformation matrix contained in `transformation.matrix` in the `metadata` slot, which is normally inferred from the header in the binary file. If there is no such matrix (e.g. if the data were streamed through a data logger that did not capture the header), `beamToXyzAdv` the user will need to store one in `x`, e.g. by doing something like the following:

```
x[["transformation.matrix"]] <- rbind(c(11100, -5771, -5321),
                                      c( #' 291, 9716, -10002),
                                      c( 1409, 1409, 1409)) / 4096
```

Author(s)

Dan Kelley

References

http://www.nortek-as.com/lib/forum-attachments/coordinate-transformation

See Also

See `read.adv` for notes on functions relating to "adv" objects.

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

---

beamUnspreadAdp *Adjust ADP Signal for Spherical Spreading*

Description

Compensate ADP signal strength for spherical spreading.

Usage

```
beamUnspreadAdp(x, count2db = c(0.45, 0.45, 0.45, 0.45), asMatrix = FALSE,
  debug = getOption("oceDebug"))
```

Arguments

`x`	An adp object, i.e. one inheriting from `adp-class`.
`count2db`	a set of coefficients, one per beam, to convert from beam echo intensity to decibels.
`asMatrix`	a boolean that indicates whether to return a numeric matrix, as opposed to returning an updated object (in which the matrix is cast to a raw value).
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

First, beam echo intensity is converted from counts to decibels, by multiplying by count2db. Then, the signal decrease owing to spherical spreading is compensated for by adding the term $20 \log 10(r)$, where r is the distance from the sensor head to the water from which scattering is occurring. r is given by x[["distance"]].

Value

An object of class "adp".

Author(s)

Dan Kelley

References

The coefficient to convert to decibels is a personal communication. The logarithmic term is explained in textbooks on acoustics, optics, etc.

See Also

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

Examples

```
library(oce)
data(adp)
plot(adp, which=5) # beam 1 echo intensity
adp.att <- beamUnspreadAdp(adp)
plot(adp.att, which=5) # beam 1 echo intensity
## Profiles
par(mar=c(4, 4, 1, 1))
a <- adp[["a", "numeric"]]             # second arg yields matrix return value
distance <- adp[["distance"]]
plot(apply(a,2,mean), distance, type='l', xlim=c(0,256))
lines(apply(a,2,median), distance, type='l',col='red')
legend("topright",lwd=1,col=c("black","red"),legend=c("original","attenuated"))
## Image
plot(adp.att, which="amplitude",col=oce.colorsJet(100))
```

binApply1D *Apply a function to vector data*

Description

The function `FUN` is applied to `f` in bins specified by `xbreaks`. (If `FUN` is `mean`, consider using `binMean2D` instead, since it should be faster.)

Usage

```
binApply1D(x, f, xbreaks, FUN, ...)
```

Arguments

`x`	a vector of numerical values.
`f`	a vector of data to which the elements of `FUN` may be supplied
`xbreaks`	values of `x` at the boundaries between bins; calculated using `pretty` if not supplied.
`FUN`	function to apply to the data
`...`	arguments to pass to the function `FUN`

Value

A list with the following elements: the breaks in x and y (`xbreaks` and `ybreaks`), the break mid-points (`xmids` and `ymids`), and a matrix containing the result of applying function `FUN` to `f` subsetted by these breaks.

Author(s)

Dan Kelley

See Also

Other bin-related functions: `binApply2D`, `binAverage`, `binCount1D`, `binCount2D`, `binMean1D`, `binMean2D`

Examples

```
library(oce)
## salinity profile with median and quartile 1 and 3
data(ctd)
p <- ctd[["pressure"]]
S <- ctd[["salinity"]]
q1 <- binApply1D(p, S, pretty(p, 30), function(x) quantile(x, 1/4))
q3 <- binApply1D(p, S, pretty(p, 30), function(x) quantile(x, 3/4))
plotProfile(ctd, "salinity", col='gray', type='n')
polygon(c(q1$result, rev(q3$result)),
c(q1$xmids, rev(q1$xmids)), col='gray')
points(S, p, pch=20)
```

binApply2D *Apply a function to matrix data*

Description

The function FUN is applied to f in bins specified by xbreaks and ybreaks. (If FUN is mean, consider using binMean2D instead, since it should be faster.)

Usage

```
binApply2D(x, y, f, xbreaks, ybreaks, FUN, ...)
```

Arguments

x	a vector of numerical values.
y	a vector of numerical values.
f	a vector of data to which the elements of FUN may be supplied
xbreaks	values of x at the boundaries between bins; calculated using pretty if not supplied.
ybreaks	values of y at the boundaries between bins; calculated using pretty if not supplied.
FUN	function to apply to the data
...	arguments to pass to the function FUN

Value

A list with the following elements: the breaks in x and y (xbreaks and ybreaks), the break midpoints (xmids and ymids), and a matrix containing the result of applying function FUN to f subsetted by these breaks.

Author(s)

Dan Kelley

See Also

Other bin-related functions: binApply1D, binAverage, binCount1D, binCount2D, binMean1D, binMean2D

Examples

```
library(oce)
## Not run:
## secchi depths in lat and lon bins
if (require(ocedata)) {
    data(secchi, package="ocedata")
```

```
    col <- rev(oce.colorsJet(100))[rescale(secchi$depth,
                                           xlow=0, xhigh=20,
                                           rlow=1, rhigh=100)]
    zlim <- c(0, 20)
    breaksPalette <- seq(min(zlim), max(zlim), 1)
    colPalette <- rev(oce.colorsJet(length(breaksPalette)-1))
    drawPalette(zlim, "Secchi Depth", breaksPalette, colPalette)
    data(coastlineWorld)
    mapPlot(coastlineWorld, longitudelim=c(-5, 20), latitudelim=c(50, 66),
      grid=5, fill='gray', projection="+proj=lcc +lat_1=50 +lat_2=65")
    bc <- binApply2D(secchi$longitude, secchi$latitude,
                     pretty(secchi$longitude, 80),
                     pretty(secchi$latitude, 40),
                     f=secchi$depth, FUN=mean)
    mapImage(bc$xmids, bc$ymids, bc$result, zlim=zlim, col=colPalette)
    mapPolygon(coastlineWorld, col='gray')
}

## End(Not run)
```

`binAverage` *Bin-average a vector y, based on x values*

Description

The y vector is averaged in bins defined for x. Missing values in y are ignored.

Usage

```
binAverage(x, y, xmin, xmax, xinc)
```

Arguments

x	a vector of numerical values.
y	a vector of numerical values.
xmin	x value at the lower limit of first bin; the minimum x will be used if this is not provided.
xmax	x value at the upper limit of last bin; the maximum x will be used if this is not provided.
xinc	width of bins, in terms of x value; 1/10th of xmax-xmin will be used if this is not provided.

Value

A list with two elements: x, the mid-points of the bins, and y, the average y value in the bins.

Author(s)

Dan Kelley

See Also

Other bin-related functions: binApply1D, binApply2D, binCount1D, binCount2D, binMean1D, binMean2D

Examples

```
library(oce)
## A. fake linear data
x <- seq(0, 100, 1)
y <- 1 + 2 * x
plot(x, y, pch=1)
ba <- binAverage(x, y)
points(ba$x, ba$y, pch=3, col='red', cex=3)

## B. fake quadratic data
y <- 1 + x ^2
plot(x, y, pch=1)
ba <- binAverage(x, y)
points(ba$x, ba$y, pch=3, col='red', cex=3)

## C. natural data
data(co2)
plot(co2)
avg <- binAverage(time(co2), co2, 1950, 2000, 2)
points(avg$x, avg$y, col='red')
```

---

binCount1D *Bin-count vector data*

Description

Count the number of elements of a given vector that fall within successive pairs of values within a second vector.

Usage

```
binCount1D(x, xbreaks)
```

Arguments

x	Vector of numerical values.
xbreaks	Vector of values of x at the boundaries between bins, calculated using pretty if not supplied.

Value

A list with the following elements: the breaks (xbreaks, midpoints (xmids) between those breaks, and the count (number) of x values between successive breaks.

Author(s)

Dan Kelley

See Also

Other bin-related functions: binApply1D, binApply2D, binAverage, binCount2D, binMean1D, binMean2D

---

binCount2D *Bin-count matrix data*

Description

Count the number of elements of a given matrix z=z(x,y) that fall within successive pairs of breaks in x and y.

Usage

```
binCount2D(x, y, xbreaks, ybreaks, flatten = FALSE)
```

Arguments

x	Vector of numerical values.
y	Vector of numerical values.
xbreaks	Vector of values of x at the boundaries between bins, calculated using pretty(x) if not supplied.
ybreaks	Vector of values of y at the boundaries between bins, calculated using pretty(y) if not supplied.
flatten	A logical value indicating whether the return value also contains equilength vectors x, y, z and n, a flattened representation of xmids, ymids, result and number.

Value

A list with the following elements: the breaks (xbreaks and ybreaks), the midpoints (xmids and ymids) between those breaks, and the count (number) of f values in the boxes defined between successive breaks.

Author(s)

Dan Kelley

See Also

Other bin-related functions: binApply1D, binApply2D, binAverage, binCount1D, binMean1D, binMean2D binmapAdp *Bin-map an ADP object*

Description

Bin-map an ADP object, by interpolating velocities, backscatter amplitudes, etc., to uniform depth bins, thus compensating for the pitch and roll of the instrument. This only makes sense for ADP objects that are in beam coordinates.

Usage

```
binmapAdp(x, debug = getOption("oceDebug"))
```

Arguments

`x`	an adp object, i.e. one inheriting from `adp-class`.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Value

An object of `class` "adp".

Bugs

This only works for 4-beam RDI ADP objects.

Author(s)

Dan Kelley and Clark Richards

References

The method was devised by Clark Richards for use in his PhD work at Department of Oceanography at Dalhousie University.

See Also

See `adp-class` for a discussion of adp objects and notes on the many functions dealing with them.

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

Examples

```
## Not run:
library(oce)
beam <- read.oce("/data/archive/sleiwex/2008/moorings/m09/adp/rdi_2615/raw/adp_rdi_2615.000",
                 from=as.POSIXct("2008-06-26", tz="UTC"),
                 to=as.POSIXct("2008-06-26 00:10:00", tz="UTC"),
                 longitude=-69.73433, latitude=47.88126)
beam2 <- binmapAdp(beam)
plot(enuToOther(toEnu(beam), heading=-31.5))
plot(enuToOther(toEnu(beam2), heading=-31.5))
plot(beam, which=5:8) # backscatter amplitude
plot(beam2, which=5:8)

## End(Not run)
```

`binMean1D` *Bin-average f=f(x)*

Description

Average the values of a vector `f` in bins defined on another vector `x`. A common example might be averaging CTD profile data into pressure bins (see "Examples").

Usage

```
binMean1D(x, f, xbreaks)
```

Arguments

`x`	Vector of numerical values.
`f`	Vector of numerical values.
`xbreaks`	Vector of values of x at the boundaries between bins, calculated using `pretty` if not supplied.

Value

A list with the following elements: the breaks (`xbreaks`, midpoints (`xmids`) between those breaks, the count (`number`) of `x` values between successive breaks, and the resultant average (`result`) of `f`, classified by the `x` breaks.

Author(s)

Dan Kelley

See Also

Other bin-related functions: binApply1D, binApply2D, binAverage, binCount1D, binCount2D, binMean2D

Examples

```
library(oce)
data(ctd)
z <- ctd[["z"]]
T <- ctd[["temperature"]]
plot(T, z)
TT <- binMean1D(z, T, seq(-100, 0, 1))
lines(TT$result, TT$xmids, col='red')
```

---

binMean2D *Bin-average f=f(x,y)*

Description

Average the values of a vector `f(x,y)` in bins defined on vectors `x` and `y`. A common example might be averaging spatial data into location bins.

Usage

```
binMean2D(x, y, f, xbreaks, ybreaks, flatten = FALSE, fill = FALSE,
  fillgap = -1)
```

Arguments

x	Vector of numerical values.
y	Vector of numerical values.
f	Matrix of numerical values, a matrix f=f(x,y).
xbreaks	Vector of values of x at the boundaries between bins, calculated using pretty(x) if not supplied.
ybreaks	Vector of values of y at the boundaries between bins, calculated using pretty(y) if not supplied.
flatten	A logical value indicating whether the return value also contains equilength vectors x, y, z and n, a flattened representation of xmids, ymids, result and number.
fill	Logical value indicating whether to fill NA-value gaps in the matrix. Gaps will be filled as the average of linear interpolations across rows and columns. See fillgap, which works together with this.
fillgap	Integer controlling the size of gap that can be filled across. If this is negative (as in the default), gaps will be filled regardless of their size. If it is positive, then gaps exceeding this number of indices will not be filled.

Value

A list with the following elements: the midpoints (renamed as `x` and `y`), the count (`number`) of `f(x,y)` values for `x` and `y` values that lie between corresponding breaks, and the resultant average (`f`) of `f(x,y)`, classified by the `x` and `y` breaks.

Author(s)

Dan Kelley

See Also

Other bin-related functions: `binApply1D`, `binApply2D`, `binAverage`, `binCount1D`, `binCount2D`, `binMean1D`

Examples

```
library(oce)
x <- runif(500)
y <- runif(500)
f <- x + y
xb <- seq(0, 1, 0.1)
yb <- seq(0, 1, 0.2)
m <- binMean2D(x, y, f, xb, yb)
plot(x, y)
contour(m$xmids, m$ymids, m$result, add=TRUE, levels=seq(0, 2, 0.5), labcex=1)
```

---

bound125 — *Calculate a rounded bound, rounded up to matissa 1, 2, or 5*

Description

Calculate a rounded bound, rounded up to matissa 1, 2, or 5

Usage

```
bound125(x)
```

Arguments

`x`	a single positive number

Value

for positive x, a value exceeding x that has mantissa 1, 2, or 5; otherwise, x

bremen-class	*Class for data stored in a format used at Bremen*

Description

Class for data stored in a format used at Bremen. This is somewhat unusual amongst oce classes, in that it does not map to a particular instrument. Although some functions are provided for dealing with these data (see "Details"), the most common action is to read the data with `read.bremen`, and then to coerce the object to another storage class (e.g. using `as.ctd` for CTD-style data) so that specialized functions can be used thereafter.

The main function is `read.bremen`. A simple plotting method is provided with `plot,bremen-method`, and `summary,bremen-method` provides summaries. Data may be retrieved with `[[,bremen-method` or replaced with `[[<-,bremen-method`.

Author(s)

Dan Kelley

See Also

Other classes provided by oce: `adp-class`, `adv-class`, `argo-class`, `cm-class`, `coastline-class`, `ctd-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `oce-class`, `odf-class`, `rsk-class`, `sealevel-class`, `section-class`, `topo-class`, `windrose-class`

Other things related to bremen data: `[[,bremen-method`, `[[<-,bremen-method`, `plot,bremen-method`, `read.bremen`, `summary,bremen-method`

byteToBinary	*Format bytes as binary*

Description

WARNING: The `endian` argument will soon be removed from this function; see oce-deprecated. This is because the actions for `endian="little"` made no sense in practical work. The default value for `endian` was changed to `"big"` on 2017 May 6.

Usage

```
byteToBinary(x, endian)
```

Arguments

`x`	an integer to be interpreted as a byte.
`endian`	character string indicating the endian-ness ("big" or "little"). **This argument will be removed in the upcoming CRAN release.**

Value

A character string representing the bit strings for the elements of `x`, in order of significance for the `endian="big"` case. (The nibbles, or 4-bit sequences, are interchanged in the now-deprecated `"little"` case.) See "Examples" for how this relates to the output from rawToBits.

Author(s)

Dan Kelley

Examples

```
library(oce)
## Note comparison with rawToBits():
a <- as.raw(0x0a)
byteToBinary(a, "big") # "00001010"
rev(rawToBits(a))      # 00 00 00 00 01 00 01 00
```

---

cm   *A CM Record*

Description

The result of using `read.cm` on a current meter file holding measurements made with an InterOcean S4 device. See `read.cm` for some general cautionary notes on reading such files. Note that the salinities in this sample dataset are known to be incorrect, perhaps owing to a lack of calibration of an old instrument that had not been used in a long time.

Usage

```
data(cm)
```

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to cm data: `[[,cm-method`, `[[<-,cm-method`, `as.cm`, `cm-class`, `plot,cm-method`, `read.cm`, `subset,cm-method`, `summary,cm-method`

Examples

```
## Not run:
library(oce)
data(cm)
summary(cm)
plot(cm)

## End(Not run)
```

cm-class *Class to Store Current Meter (CM) Data*

Description

Class to store current meter data, e.g. from an Interocean/S4 device or an Aanderaa/RCM device. A file containing Interocean/S4 data may be read with `read.cm`. Alternatively, `as.cm` can be used to create `cm` objects. Objects of this class can be plotted with `plot,cm-method` or summarized with `summary,cm-method`. Data may be retrieved with `[[,cm-method` or replaced with \ `[[<-,cm-method`.

Author(s)

Dan Kelley

See Also

Other things related to cm data: `[[,cm-method`, `[[<-,cm-method`, `as.cm`, `cm`, `plot,cm-method`, `read.cm`, `subset,cm-method`, `summary,cm-method`

Other classes provided by oce: `adp-class`, `adv-class`, `argo-class`, `bremen-class`, `coastline-class`, `ctd-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `oce-class`, `odf-class`, `rsk-class`, `sealevel-class`, `section-class`, `topo-class`, `windrose-class`

cnvName2oceName *Infer variable name, units and scale from a Seabird (.cnv) header line*

Description

This function is used by `read.ctd.sbe` to infer data names and units from the coding used by Teledyne/Seabird (SBE) `.cnv` files. Lacking access to documentation on the SBE format, the present function is based on inspection of a suite of CNV files available to the oce developers.

Usage

```
cnvName2oceName(h, columns = NULL, debug = getOption("oceDebug"))
```

Arguments

h	The header line.
columns	Optional list containing name correspondances, as described for `read.ctd.sbe`.
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

A few sample header lines that have been encountered are:

```
# name 4 = t068: temperature, IPTS-68 [deg C]
# name 3 = t090C: Temperature [ITS-90, deg C]
# name 4 = t190C: Temperature, 2 [ITS-90, deg C]
```

Examination of several CNV files suggests that it is best to try to infer the name from the characters between the "=" and ":" characters, because the material after the colon seems to vary more between sample files.

The table given below indicates the translation patterns used. These are taken from [1]. The `.cnv` convention for multiple sensors is to include optional extra digits in the name, and these are indicated with ~ in the table; their decoding is done with `grep`.

It is important to note that this table is by no means complete, since there are a great many SBE names listed in their document [1], plus names not listed there but present in data files supplied by prominent archiving agencies. If an SBE name is not recognized, then the oce name is set to that SBE name. This can cause problems in some other processing steps (e.g. if `swRho` or a similar function is called with an oce object as first argument), and so users are well-advised to rename the items as appropriate. The first step in doing this is to pass the object to `summary()`, to discover the SBE names in question. Then consult the SBE documentation to find an appropriate name for the data, and either manipulate the names in the object data slot directly or use `renameData` to rename the elements. Finally, please publish an 'issue' on the oce Github site https://github.com/dankelley/oce/issues so that the developers can add the data type in question. (To save development time, there is no plan to add all possible data types without a reasonable and specific expression user interest. Oxygen alone has over forty variants.)

Key	Result	Unit;scale	Notes
`alt`	`altimeter`	m	
`altM`	`altimeter`	m	
`accM`	`acceleration`	m/s^2	
`bat~`	`beamAttenuation`	1/m	
`C2-C1S/m`	`conductivityDifference`	S/m	
`C2-C1mS/cm`	`conductivityDifference`	mS/cm	
`C2-C1uS/cm`	`conductivityDifference`	uS/cm	
`c~mS/cm`	`conductivity`	mS/cm	
`c~S/m`	`conductivity`	S/m	
`c~uS/cm`	`conductivity`	uS/cm	
`CStarAt~`	`beamAttenuation`	1/m	
`CStarTr~`	`beamTransmission`	percent	
`density~~`	`density`	kg/m^3	
`depS`	`depth`	m	
`depSM`	`depth`	m	
`depF`	`depth`	m	
`depFM`	`depth`	m	
`dz/dtM`	`descentRate`	m/s	
`f~`	`frequency`	Hz	
`f~~`	`frequency`	Hz	
`flC~`	`fluorescence`	ug/l; Chelsea Aqua 3	

flCM	fluorescence	ug/l; Chelsea Mini Chl Con	
flCUVA~	fluorescence	ug/l; Chelsea UV Aquatracka	
flEC-AFL~	fluorescence	mg/m^3; WET Labs ECO-AFL/FLtab	
flS	fluorescence	-; Seatech	
flScufa~	fluorescence	-; Turner SCUFA [RFU]	
flSP	fluorescence	-; Seapoint	
flSPR	fluorescence	-; Seapoint, Rhodamine	
flSPuv	fluorescence	-; Seapoint, UV	
flT	fluorescence	-; Turner 10-005 flT	
gpa	geopotentialAnomaly	-; J/kg	
latitude	latitude	degN	
longitude	longitude	degE	
n2satML/L	nitrogenSaturation	ml/l	
n2satMg/L	nitrogenSaturation	mg/l	
n2satumol/kg	nitrogenSaturation	umol/kg	
nbin	nbin		
obsscufa~	backscatter	NTU; Turner SCUFA	
opoxMg/L	oxygen	mg/l; Optode, Anderaa	
opoxML/L	oxygen	ml/l; Optode, Anderaa	
opoxMm/L	oxygen	umol/l; Optode, Anderaa	
opoxPS	oxygen	percent; Optode, Anderaa	
oxsatML/L	oxygen	ml/l; Weiss	
oxsatMg/L	oxygen	mg/l; Weiss	
oxsatMm/Kg	oxygen	umol/kg; Weiss	
oxsolML/L	oxygen	ml/l; Garcia-Gordon	
oxsolMg/L	oxygen	mg/l; Garcia-Gordon	
oxsolMm/Kg	oxygen	umol/kg; Garcia-Gordon	
par~	PAR	-; Biospherical/Licor	
par/log	PAR	log; Satlantic	
ph	pH	-	
potemp~68C	thetaM	degC; IPTS-68	
potemp~90C	thetaM	degC; ITS-90	
pr	pressure	dbar	1
prM	pressure	dbar	
pr50M	pressure	dbar; SBE50	
prSM	pressure	dbar	
prDM	pressure	dbar; digiquartz	
prdE	pressure	psi; strain gauge	2
prDE	pressure	psi; digiquartz	2
prdM	pressure	dbar; strain gauge	
prSM	pressure	dbar; strain gauge	
ptempC	pressureTemperature	degC; ITS-90	3
pumps	pumpStatus		
rhodflTC~	Rhodamine	ppb; Turner Cyclops	
sal~~	salinity	-, PSS-78	4
sbeox~ML/L	oxygen	ml/l; SBE43	
sbeox~Mm/Kg	oxygen	umol/kg; SBE43	
sbeox~Mm/L	oxygen	umol/l; SBE43	

sbeox~PS	oxygen	percent; SBE43	
sbeox~V	oxygenRaw	V; SBE43	
scan	scan	-	
seaTurbMtr~	turbidity	FTU; SeaPoint	
secS-priS	salinityDifference	-, PSS-78	
sigma-t	sigmaT	kg/m^3	
sigma-theta	sigmaTheta	kg/m^3	5
sigma-é	sigmaTheta	kg/m^3	5
spar	spar	-	
specc	conductivity	uS/cm	
sva	specificVolumeAnomaly	1e-8 m^3/kg;	
svCM~	soundSpeed	m/s; Chen-Millero	
T2~68C	temperatureDifference	degC; IPTS-68	
T2~90C	temperatureDifference	degC; ITS-90	
t~68	temperature	degC; IPTS-68	
t~90	temperature	degC; ITS-90	
t~68	temperature	degC; IPTS-68	
t~68C	temperature	degC; IPTS-68	
t~90C	temperature	degC; ITS-90	
t090Cm	temperature	degC; ITS-90	
t4990C	temperature	degC; ITS-90	
tnc90C	temperature	degC; ITS-90	
tsa	thermostericAnomaly	1e-8 m^3/kg	
tv290C	temperature	degC; ITS-90	
t4968C	temperature	degC; IPTS-68	
tnc68C	temperature	degC; IPTS-68	
tv268C	temperature	degC; IPTS-68	
t190C	temperature	degC; ITS-90	
tnc290C	temperature	degC; ITS-90	
tnc268C	temperature	degC; IPTS-68	
t3890C~	temperature	degC; ITS-90	
t38~90C	temperature	degC; ITS-90	
t3868C~	temperature	degC; IPTS-68	
t38~38C	temperature	degC; IPTS-68	
timeH	time	hour; elapsed	
timeJ	time	day; elapsed	
timeK	time	s; since Jan 1, 2000	
timeM	time	minute; elapsed	
timeN	time	s; NMEA since Jan 1, 1970	
timeQ	time	s; NMEA since Jan 1, 2000	
timeS	time	s; elapsed	
turbflTC~	turbidity	NTU; Turner Cyclops	
turbflTCdiff	turbidityDifference	NTU; Turner Cyclops	
turbWETbb~	turbidity	1/(m*sr); WET Labs ECO	
turbWETbbdiff	turbidityDifference	1/(m*sr); WET Labs ECO	
turbWETntu~	turbidity	NTU; WET Labs ECO	
turbWETntudiff	turbidityDifference	NTU; WET Labs ECO	
upoly~	upoly	-	

user~	user	-
v~~	voltage	V
wetBAttn	beamAttenuation	1/m; WET Labs AC3
wetBTrans	beamTransmission	percent; WET Labs AC3
wetCDOM~	fluorescence	mg/m^3; WET Labs CDOM
wetCDOMdiff	fluorescenceDifference	mg/m^3; WET Labs CDOM
wetChAbs	fluorescence	1/m; WET Labs AC3 absorption
wetStar~	fluorescence	mg/m^3; WET Labs WETstar
wetStardiff	fluorescenceDifference	mg/m^3; WET Labs WETstar
xmiss	beamTransmission	percent; Chelsea/Seatech
xmiss~	beamTransmission	percent; Chelsea/Seatech

Notes:

- 1: 'pr' is in a Dalhousie-generated data file but seems not to be in [1].
- 2: this is an odd unit, and so if `sw*` functions are called on an object containing this, a conversion will be made before performing the computation. Be on the lookout for errors, since this is a rare situation.
- 3: assume ITS-90 temperature scale, since sample `.cnv` file headers do not specify it.
- 4: some files have PSU for this. Should we handle that? And are there other S scales to consider?
- 5: 'theta' may appear in different ways with different encoding configurations, set up within R or in the operating system.

Value

a list containing `name` (the oce name), `nameOriginal` (the SBE name) and `unit`.

Author(s)

Dan Kelley

References

1. A SBE data processing manual is at `http://www.seabird.com/document/sbe-data-processing-manual`.

See Also

Other things related to ctd data: `[[,ctd-method`, `[[<-,ctd-method`, `as.ctd`, `ctd-class`, `ctdDecimate`, `ctdFindProfiles`, `ctdRaw`, `ctdTrim`, `ctd`, `handleFlags,ctd-method`, `oceNames2whpNames`, `oceUnits2whpUnits`, `plot,ctd-method`, `plotProfile`, `plotScan`, `plotTS`, `read.ctd.itp`, `read.ctd.odf`, `read.ctd.sbe`, `read.ctd.woce.other`, `read.ctd.woce`, `read.ctd`, `subset,ctd-method`, `summary,ctd-method`, `woceNames2oceNames`, `woceUnit2oceUnit`, `write.ctd`

Other functions that interpret variable names and units from headers: `ODFNames2oceNames`, `oceNames2whpNames`, `oceUnits2whpUnits`, `unitFromStringRsk`, `unitFromString`, `woceNames2oceNames`, `woceUnit2oceUnit` coastline-class *Class to Store Coastline Data*

Description

Class to store coastline data, which may be read with read.coastline or constructed with as.coastline, plotted with plot,coastline-method or summarized with summary,coastline-method. Data within coastline objects may be retrieved with [[,coastline-method or replaced with [[<-,coastline-method.

Author(s)

Dan Kelley

See Also

Other classes provided by oce: adp-class, adv-class, argo-class, bremen-class, cm-class, ctd-class, echosounder-class, lisst-class, lobo-class, met-class, oce-class, odf-class, rsk-class, sealevel-class, section-class, topo-class, windrose-class

Other things related to coastline data: [[,coastline-method, [[<-,coastline-method, as.coastline, coastlineBest, coastlineCut, coastlineWorld, download.coastline, plot,coastline-method, read.coastline.openstreetmap, read.coastline.shapefile, subset,coastline-method, summary,coastline-meth

coastlineBest *Find the Name of the Best Coastline Object*

Description

Find the name of the most appropriate coastline for a given locale Checks coastlineWorld, coastlineWorldFine and coastlineWorldCoarse, in that order, to find the one most appropriate for the locale.

Usage

```
coastlineBest(lonRange, latRange, span, debug = getOption("oceDebug"))
```

Arguments

lonRange	range of longitude for locale
latRange	range of latitude for locale
span	span of domain in km (if provided, previous two arguments are ignored).
debug	set to a positive value to get debugging information during processing.

Value

The name of a coastline that can be loaded with data().

Author(s)

Dan Kelley

See Also

Other things related to coastline data: [[,coastline-method, [[<-,coastline-method, as.coastline, coastline-class, coastlineCut, coastlineWorld, download.coastline, plot,coastline-method, read.coastline.openstreetmap, read.coastline.shapefile, subset,coastline-method, summary,coastline-meth

---

coastlineCut *Cut a Coastline Object at Specified Longitude*

Description

This can be helpful in preventing mapPlot from producing ugly horizontal lines in world maps. These lines occur when a coastline segment is intersected by longitude lon_0+180. Since the coastline files in the oce and ocedata packages are already "cut" at longitudes of -180 and 180, the present function is not needed for default maps, which have `+lon_0=0`. However, may help with other values of `lon_0`.

Usage

```
coastlineCut(coastline, lon_0 = 0)
```

Arguments

`coastline`	original coastline object
`lon_0`	longitude as would be given in a `+lon_0=` item in a call to project in the **rgdal** package.

Value

a new coastline object

Caution

This function is provisional. Its behaviour, name and very existence may change. Part of the development plan is to see if there is common ground between this and the `clipPolys` function in the **PBSmapping** package.

See Also

Other things related to coastline data: [[,coastline-method, [[<-,coastline-method, as.coastline, coastline-class, coastlineBest, coastlineWorld, download.coastline, plot,coastline-method, read.coastline.openstreetmap, read.coastline.shapefile, subset,coastline-method, summary,coastline-meth

Examples

```
library(oce)
data(coastlineWorld)
## Not run:
mapPlot(coastlineCut(coastlineWorld, lon_0=100), proj="+proj=robin +lon_0=100")#, col='gray')

## End(Not run)
```

---

coastlineWorld *World Coastline*

Description

This is a coarse resolution coastline at scale 1:110M, with 10,696 points, suitable for world-scale plots plotted at a small size, e.g. inset diagrams. Finer resolution coastline files are provided in the `ocedata` package.

Installing your own datasets

Follow the procedure along the lines described in "Details", where of course your source file will differ. Also, you should change the name of the coastline object from `coastlineWorld`, to avoid conflicts with the built-in dataset. Save the `.rda` file to some directory of your choosing, e.g. perhaps `/data/coastlines` or `~/data/coastlines` on a unix-type machine. Then, whenever you need the file, use `load` to load it. Most users find it convenient to do the loading in an `Rprofile` startup file.

Author(s)

Dan Kelley

Source

Downloaded from `http://www.naturalearthdata.com`, in `ne_110m_admin_0_countries.shp` in July 2015, with an update on December 16, 2017.

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to coastline data: `[[,coastline-method`, `[[<-,coastline-method`, `as.coastline`, `coastline-class`, `coastlineBest`, `coastlineCut`, `download.coastline`, `plot,coastline-method`, `read.coastline.openstreetmap`, `read.coastline.shapefile`, `subset,coastline-method`, `summary,coastline-meth` colormap *Calculate colour map*

Description

Map values to colours, for use in palettes and plots. There are many ways to use this function, and some study of the arguments should prove fruitful in cases that extend far beyond the examples.

Usage

```
colormap(z = NULL, zlim, zclip = FALSE, breaks, col = oce.colorsJet, name,
  x0, x1, col0, col1, blend = 0, missingColor,
  debug = getOption("oceDebug"))
```

Arguments

`z`	an optional vector or other set of numerical values to be examined. If `z` is given, the return value will contain an item named `zcol` that will be a vector of the same length as `z`, containing a colour for each point. If `z` is not given, `zcol` will contain just one item, the colour `"black"`.
`zlim`	optional vector containing two numbers that specify the `z` limits for the colour scale. If provided, it overrides defaults as describe in the following. If `name` is given, then the `range` of numerical values contained therein will be used for `zlim`. Otherwise, if `z` is given, then its `rangeExtended` sets `zlim`. Otherwise, if `x0` and `x1` are given, then their `range` sets `zlim`. Otherwise, there is no way to infer `zlim` and indeed there is no way to construct a colormap, so an error is reported. It is an error to specify both `zlim` and `breaks`, if the length of the latter does not equal 1.
`zclip`	logical, with `TRUE` indicating that `z` values outside the range of `zlim` or `breaks` should be painted with `missingColor` and `FALSE` indicating that these values should be painted with the nearest in-range colour.
`breaks`	an optional indication of break points between colour levels (see `image`). If this is provided, the arguments `name` through `blend` are all ignored (see "Details"). If it is provided, then it may either be a vector of break points, or a single number indicating the desired number of break points to be computed with `pretty(z, breaks)`. In either case of non-missing `breaks`, the resultant break points must number 1 plus the number of colours (see `col`).
`col`	either a vector of colours or a function taking a numerical value as its single argument and returning a vector of colours. The value of `col` is ignored if `name` is provided, or if `x0` through `col1` are provided.
`name`	an optional string naming a built-in colormap (one of `"gmt_relief"`, `"gmt_ocean"`, `"gmt_globe"` or `"gmt_gebco"`) or the name of a file or URL that contains a colour map specification in GMT format, e.g. one of the `.cpt` files from `http://www.beamreach.org/maps/gmt/share/cpt`). If `name` is provided, then `x0`, `x1`, `col0` and `col1` are all ignored.

`x0, x1, col0, col1`
: Vectors that specify a colour map. They must all be the same length, with `x0` and `x1` being numerical values, and `col0` and `col1` being colours. The colours may be strings (e.g. `"red"`) or colours as defined by `rgb` or `hsv`.

`blend`
: a number indicating how to blend colours within each band. This is ignored except when `x0` through `col1` are supplied. A value of 0 means to use `col0[i]` through the interval `x0[i]` to `x1[i]`. A value of 1 means to use `col1[i]` in that interval. A value between 0 and 1 means to blend between the two colours according to the stated fraction. Values exceeding 1 are an error at present, but there is a plan to use this to indicate subintervals, so a smooth palette can be created from a few colours.

`missingColor`
: colour to use for missing values. If not provided, this will be `"gray"`, unless `name` is given, in which case it comes from that colour table.

`debug`
: a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

This is a multi-purpose function that generally links ("maps") numerical values to colours. The return value can specify colours for points on a graph, or `breaks` and `col` vectors that are suitable for use by `drawPalette`, `imagep` or `image`.

There are three ways of specifying colour schemes, and `colormap` works by checking for each condition in turn.

- Case A. Supply `z` but nothing else. In this case, `breaks` will be set to `pretty(z, 10)` and things are otherwise as in case B.
- Case B. Supply `breaks`. In this case, `breaks` and `col` are used together to specify a colour scheme. If `col` is a function, then it is expected to take a single numerical argument that specifies the number of colours, and this number will be set to `length(breaks)-1`. Otherwise, `col` may be a vector of colours, and its length must be one less than the number of breaks. (NB. if `breaks` is given, then all other arguments except `col` and `missingColor` are ignored.)
- Case C. Do not supply `breaks`, but supply `name` instead. This `name` may be the name of a pre-defined colour palette (`"gmt_relief"`, `"gmt_ocean"`, `"gmt_globe"` or `"gmt_gebco"`), or it may be the name of a file (including a URL) containing a colour map in the GMT format (see "References"). (NB. if `name` is given, then all other arguments except `z` and `missingColor` are ignored.)
- Case D. Do not supply either `breaks` or `name`, but instead supply each of `x0`, `x1`, `col0`, and `col1`. These values are specify a value-colour mapping that is similar to that used for GMT colour maps. The method works by using `seq` to interpolate between the elements of the `x0` vector. The same is done for `x1`. Similarly, `colorRampPalette` is used to interpolate between the colours in the `col0` vector, and the same is done for `col1`.

Value

A list containing the following (not necessarily in this order)

- `zcol`, a vector of colours for `z`, if `z` was provided, otherwise `"black"`

- zlim, a two-element vector suitable as the argument of the same name supplied to image or imagep
- breaks and col, vectors of breakpoints and colours, suitable as the same-named arguments to image or imagep
- zclip the provided value of zclip.
- x0 and x1, numerical vectors of the sides of colour intervals, and col0 and col1, vectors of corresponding colours. The meaning is the same as on input. The purpose of returning these four vectors is to permit users to alter colour mapping, as in example 3 in "Examples".
- missingColor, a colour that could be used to specify missing values, e.g. as the same-named argument to imagep. If this is supplied as an argument, its value is repeated in the return value. Otherwise, its value is either "gray" or, in the case of name being given, the value in the GMT colour map specification.

Author(s)

Dan Kelley

References

Information on GMT software is given at http://gmt.soest.hawaii.edu (link worked for years but failed 2015-12-12). Diagrams showing the GMT colour schemes are at http://www.geos.ed.ac.uk/it/howto/GMT/CP (link worked for years but failed 2015-12-08), and numerical specifications for some colour maps are at http://www.beamreach.org/maps/gmt/share/cpt, http://soliton.vm.bytemark.co.uk/pub/cpt-city, and other sources.

See Also

Other things related to colors: colors, oce.colorsGebco, oce.colorsTwo

Examples

```
library(oce)
## Example 1. colour scheme for points on xy plot
x <- seq(0, 1, length.out=40)
y <- sin(2 * pi * x)
par(mar=c(3, 3, 1, 1))
mar <- par('mar') # prevent margin creep by drawPalette()
## First, default breaks
c <- colormap(y)
drawPalette(c$zlim, col=c$col, breaks=c$breaks)
plot(x, y, bg=c$zcol, pch=21, cex=1)
grid()
par(mar=mar)
## Second, 100 breaks, yielding a smoother palette
c <- colormap(y, breaks=100)
drawPalette(c$zlim, col=c$col, breaks=c$breaks)
plot(x, y, bg=c$zcol, pch=21, cex=1)
grid()
par(mar=mar)
```

```
## Not run:
## Example 2. topographic image with a standard colour scheme
par(mfrow=c(1,1))
data(topoWorld)
cm <- colormap(name="gmt_globe")
imagep(topoWorld, breaks=cm$breaks, col=cm$col)

## Example 3. topographic image with modified colours,
## black for depths below 4km.
cm <- colormap(name="gmt_globe")
deep <- cm$x0 < -4000
cm$col0[deep] <- 'black'
cm$col1[deep] <- 'black'
cm <- colormap(x0=cm$x0, x1=cm$x1, col0=cm$col0, col1=cm$col1)
imagep(topoWorld, breaks=cm$breaks, col=cm$col)

## Example 4. image of world topography with water colorized
## smoothly from violet at 8km depth to blue
## at 4km depth, then blending in 0.5km increments
## to white at the coast, with tan for land.
cm <- colormap(x0=c(-8000, -4000,   0,  100),
               x1=c(-4000,     0, 100, 5000),
               col0=c("violet","blue","white","tan"),
               col1=c("blue","white","tan","yelloe"),
               blend=c(100, 8, 0))
lon <- topoWorld[['longitude']]
lat <- topoWorld[['latitude']]
z <- topoWorld[['z']]
imagep(lon, lat, z, breaks=cm$breaks, col=cm$col)
contour(lon, lat, z, levels=0, add=TRUE)
message("colormap() example 4 is broken")

## Example 5. visualize GMT style colour map
cm <- colormap(name="gmt_globe", debug=4)
plot(seq_along(cm$x0), cm$x0, pch=21, bg=cm$col0)
grid()
points(seq_along(cm$x1), cm$x1, pch=21, bg=cm$col1)

## End(Not run)
```

---

colors *Data that define some colour palettes*

Description

The colors dataset is a list containing vectors of colour-scheme names, e.g. colors$viridis holds colours for the colour palette known as Viridis, which in 2015 became the default colour palette in the matplotlib 2.0 Python library [1].

Author(s)

Authored by matplotlib contributes, packaged in oce by Dan Kelley

Source

The data come from the code in matplotlib [1].

References

1. Matplotlib is developed on github; see https://github.com/matplotlib/matplotlib

See Also

`oceColorsViridis` uses this dataset.

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to colors: `colormap`, `oce.colorsGebco`, `oce.colorsTwo`

`composite` *Create a composite object by averaging across good data*

Description

Items within the `data` slots of the objects that are supplied as arguments are averaged in a way that makes sense for the object class, i.e. taking into account the particular bad-data codes of that particular class.

Usage

```
composite(object, ...)
```

Arguments

`object`	Either a `list` of oce-class objects, in which case this is the only argument, or a single oce-class object, in which case at least one other argument (an object of the same size) must be supplied.
`...`	Ignored, if `object` is a list. Otherwise, one or more `oce-class` objects of the same sub-class as the first argument.

See Also

Other functions that create composite objects: `composite,amsr-method`, `composite,list-method`

composite,amsr-method *Create a composite of amsr satellite data*

Description

Items within the data slots of the objects that are supplied as arguments are averaged in a way that makes sense for the object class, i.e. taking into account the particular bad-data codes of that particular class.

Usage

```
## S4 method for signature 'amsr'
composite(object, ...)
```

Arguments

object	An object inheriting from amsr-class.
...	Other amsr objects.

Details

Form averages for each item in the data slot of the supplied objects, taking into account the bad-data codes. If none of the objects has good data at any particular pixel (i.e. particular latitude and longitude), the resultant will have the bad-data code of the last item in the argument list. The metadata in the result are taken directly from the metadata of the final argument, except that the filename is set to a comma-separated list of the component filenames.

See Also

Other things related to amsr data: [[<-,amsr-method, amsr-class, download.amsr, plot,amsr-method, read.amsr, subset,amsr-method, summary,amsr-method

Other functions that create composite objects: composite,list-method, composite

composite,list-method *Composite by Averaging Across Data*

Description

This is done by calling a specialized version of the function defined in the given class. In the present version, the objects must inherit from amsr-class, so the action is to call composite,amsr-method.

Items within the data slots of the objects that are supplied as arguments are averaged in a way that makes sense for the object class, i.e. taking into account the particular bad-data codes of that particular class.

Usage

```
## S4 method for signature 'list'
composite(object)
```

Arguments

`object`	A `list` of oce-class objects.

See Also

Other functions that create composite objects: `composite,amsr-method`, `composite`

`coriolis` *Coriolis parameter on rotating earth*

Description

Compute f, the Coriolis parameter as a function of latitude [1], assuming earth siderial angular rotation rate $omega$=7292115e-11 rad/s. See [1] for general notes, and see [2] for comments on temporal variations of $omega$.

Usage

```
coriolis(latitude, degrees = TRUE)
```

Arguments

`latitude`	Vector of latitudes in $^\circ$N or radians north of the equator.
`degrees`	Flag indicating whether degrees are used for latitude; if set to `FALSE`, radians are used.

Value

Coriolis parameter [radian/s].

Author(s)

Dan Kelley

References

1. Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

2. Groten, E., 2004: Fundamental Parameters and Current, 2004. Best Estimates of the Parameters of Common Relevance to Astronomy, Geodesy, and Geodynamics. Journal of Geodesy, 77:724-797. (downloaded from `http://www.iag-aig.org/attach/e354a3264d1e420ea0a9920fe762f2a0/51-groten.pdf` March 11, 2017).

Examples

```
C <- coriolis(45) # 1e-4
```

---

ctd *A CTD profile in Halifax Harbour*

Description

This is a CTD profile measured in Halifax Harbour in 2003, based on ctdRaw, but trimmed to just the downcast with ctdTrim, using indices inferred by inspection of the results from plotScan.

Usage

```
data(ctd)
```

Details

This station was sampled by students enrolled in the Dan Kelley's Physical Oceanography class at Dalhousie University. The data were acquired near the centre of the Bedford Basin of the Halifax Harbour, during an October 2003 field trip of Dalhousie University's Oceanography 4120/5120 class. The original .cnv data file had temperature in the IPTS-68 scale, but this was converted to the more modern scale using T90fromT68.

See Also

The full profile (not trimmed to the downcast) is available as data(ctdRaw).

Other datasets provided with oce: adp, adv, argo, cm, coastlineWorld, colors, ctdRaw, echosounder, landsat, lisst, lobo, met, rsk, sealevelTuktoyaktuk, sealevel, section, topoWorld, wind

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
## Not run:
library(oce)
data(ctd)
plot(ctd)

## End(Not run)
```

ctd-class *Class to Store CTD (or general hydrographic) Data*

Description

Class to store hydrographic data such as measured with a CTD (conductivity, temperature, depth) instrument, or with other systems that produce similar data. Data repositories may store conductivity, temperature and depth, as in the instrument name, but it is also common to store salinity, temperature and pressure instead (or in addition). For this reason, `ctd` objects are required to hold `salinity`, `temperature` and `pressure` in their `data` slot, with other data being optional. Formulae are available for converting between variants of these data triplets, e.g. `swSCTp` can calculate `salinity` given `conductivity`, `temperature` and `pressure`, and these are used by the main functions that create `ctd` objects. For example, if `read.ctd.sbe` is used to read a Seabird file that contains only conductivity, temperature and pressure, then that function will automatically append a data item to hold salinity. `as.ctd` acts similarly. The result this is that all `ctd` objects hold `salinity`, `temperature` and `pressure`, which are henceforth called the three basic quantities.

Details

Different units and scales are permitted for the three basic quantities, and most `oce` functions check those units and scales before doing calculations (e.g. of seawater density), because those calculations demand certain units and scales. The way this is handled is that the accessor function `[[,ctd-method` returns values in standardized form. For example, a `ctd` object might hold temperature defined on the IPTS-68 scale, but e.g. `ctd[["temperature"]]` returns a value on the ITS-90 scale. (The conversion is done with `T90fromT68`.) Similarly, pressure may be stored in either dbars or PSI, but e.g. `ctd[["pressure"]]` returns a value in dbars, after multiplying by 0.689476 if the value is stored in PSI. Luckily, there is (as of early 2016) only one salinity scale in common use in data files, namely PSS-78.

The TEOS-10 notation for these quantities also works, with `ctd[["SP"]]`, `ctd[["t"]]` and `ctd[["p"]]` returning identical values to those returned for the longer names.

After the names listed above have been checked, the remaining names in the `data` slot are checked using `pmatch`, so that e.g. `ctd[["sal"]]` will recover practical `salinity`, `ctd[["sc"]]` will recover `scan` (if it exists), etc.

Accessing data

Data may be extracted with `[[,ctd-method` and inserted with `[[<-,ctd-method`. As noted above, `[[,ctd-method` returns temperature in the ITS-90 scale and pressure in dbar, regardless of the scale and unit of the data within the object. Type `?"[[,ctd-method"` or `?"[[<-,ctd-method"` to learn more.

Depth is accessed with e.g. `ctd[["depth"]]`, while its negative, the vertical coordinate, is accessed with e.g. `ctd[["z"]]`; note that these are calculated using `swDepth` and `swZ`, and that any values that may have been read in a data file are ignored.

Potential temperature defined according to UNESCO-1980 is calculated with `ctd[["theta"]]` or `ctd[["potential temperature"]]`. Salinity is retrieved with `ctd[["S"]]` or `ctd[["salinity"]]`.

Conservative Temperature defined according to TEOS-2010 is calculated with `ctd[["CT"]]` or `ctd[["conservative temperature"]]`. Absolute salinity is calculated with `ctd[["SA"]]` or `ctd[["absolute salinity"]]`. Note that the `CT`, `SA` and `Sstar` calculations require latitude and longitude, and so errors result if these items are sought for a `ctd` object that lacks latitude or longitude. (Until 2017 May 14, defaults of 300E and 30N were used if position was not stored in the object.)

The square of buoyancy frequency is retrieved with `ctd[["N2"]]` or `swN2`, density ratio with `ctd[["Rrho"]]` and spiciness with `ctd[["spice"]]`.

Extracting values

Items stored in the object may be altered with e.g. `ctd[["salinity"]] <- rep(35,10)`. For obvious reasons, this does not work with derived quantities such as conservative temperature, etc.

Reading/creating data

A file containing CTD profile data may be read with `read.ctd`, and a CTD object can also be created with `as.ctd`. See `read.ctd` for references on data formats used in CTD files. Data can also be assembled into `ctd` objects with `as.ctd`.

Statistical summaries are provided by `summary,ctd-method`, while `show` displays an overview.

CTD objects may be plotted with `plot,ctd-method`, which does much of its work by calling `plotProfile` or `plotTS`, both of which can also be called by the user, to get fine control over the plots.

A CTD profile can be isolated from a larger record with `ctdTrim`, a task made easier when `plotScan` is used to examine the results. Towyow data can be split up into sets of profiles (ascending or descending) with `ctdFindProfiles`. CTD data may be smoothed and/or cast onto specified pressure levels with `ctdDecimate`.

As with all oce objects, low-level manipulation may be done with `oceSetData` and `oceSetMetadata`. Additionally, many of the contents of CTD objects may be altered with the `[[,ctd-method` scheme discussed above, and skilled users may also manipulate the contents directly.

Author(s)

Dan Kelley

See Also

Other things related to `ctd` data: `[[,ctd-method`, `[[<-,ctd-method`, `as.ctd`, `cnvName2oceName`, `ctdDecimate`, `ctdFindProfiles`, `ctdRaw`, `ctdTrim`, `ctd`, `handleFlags,ctd-method`, `oceNames2whpNames`, `oceUnits2whpUnits`, `plot,ctd-method`, `plotProfile`, `plotScan`, `plotTS`, `read.ctd.itp`, `read.ctd.odf`, `read.ctd.sbe`, `read.ctd.woce.other`, `read.ctd.woce`, `read.ctd`, `subset,ctd-method`, `summary,ctd-method`, `woceNames2oceNames`, `woceUnit2oceUnit`, `write.ctd`

Other classes provided by oce: `adp-class`, `adv-class`, `argo-class`, `bremen-class`, `cm-class`, `coastline-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `oce-class`, `odf-class`, `rsk-class`, `sealevel-class`, `section-class`, `topo-class`, `windrose-class` ctdAddColumn *Add a Column to the Data Slot of a CTD Object [deprecated]*

Description

WARNING: This function will be removed soon; see oce-deprecated. Use `oceSetData` instead of the present function.

Usage

```
ctdAddColumn(x, column, name, label, unit = NULL, log = TRUE,
  originalName = "", debug = getOption("oceDebug"))
```

Arguments

`x`	A `ctd` object, i.e. one inheriting from `ctd-class`.
`column`	A column of data to be inserted, in the form of a numeric vector, whose length matches that of columns in the object.
`name`	Character string indicating the name this column is to have in the `data` slot of `x`.
`label`	Optional character string or expression indicating the name of the column, as it will appear in plot labels. (If not given, `name` will be used.)
`unit`	Optional indication of the unit, in the form of a list containing items `unit`, which is an expression, and `scale`, which is a character string. For example, modern measurements of temperature have unit `list(name=expression(degree*C), scale="ITS-90")`.
`log`	A logical value indicating whether to store an entry in the processing log that indicates this insertion.
`originalName`	string indicating the name of the data element as it was originally. This makes sense only for data being read from a file, where e.g. WOCE or SBE names might be used.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

Add a column to the `data` slot of an object of `ctd-class`, also updating the `metadata` slot as appropriate.

Value

A ctd object.

Author(s)

Dan Kelley

See Also

The documentation for ctd-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Other functions that will be removed soon: addColumn, ctdUpdateHeader, mapMeridians, mapZones, oce.as.POSIXlt

Examples

```
library(oce)
data(ctd)
F <- 32 + (9/5)*ctd[["temperature"]]
ctdNew <- ctdAddColumn(ctd, F, "temperatureF",
    unit=list(unit=expression(degree*F), scale="ITS-90"))
```

---

ctdDecimate *Decimate a CTD profile*

Description

Interpolate a CTD profile to specified pressure values.

Usage

```
ctdDecimate(x, p = 1, method = "boxcar", rule = 1, e = 1.5,
  debug = getOption("oceDebug"))
```

Arguments

x	A ctd object, i.e. one inheriting from ctd-class.
p	pressure increment, or vector of pressures. In the first case, pressures from 0dbar to the rounded maximum pressure are used, incrementing by p dbars. If a vector of pressures is given, interpolation is done to these pressures.
method	the method to be used for calculating decimated values. This may be a function or a string naming a built-in method. The built-in methods are "boxcar" (based on a local average), "approx" (based on linear interpolation between neighboring points, using approx with the rule argument specified here), "lm" (based on local regression, with e setting the size of the local region), "rr" (for the Reiniger and Ross method, carried out with oce.approx) and "unesco" (for the UNESCO method, carried out with. oce.approx. If method is a function, then it must take three arguments, the first being pressure, the second being an arbitrary variable in another column of the data, and the third being a vector of target pressures at which the calculation is carried out, and the return value must be a vector. See "Examples".

rule	an integer that is passed to approx, in the case where method is "approx". Note that the default value for rule is 1, which will inhibit extrapolation beyond the observed pressure range. This is a change from the behaviour previous to May 8, 2017, when a rule of 2 was used (without stating so as an argument).
e	is an expansion coefficient used to calculate the local neighbourhoods for the "boxcar" and "lm" methods. If e=1, then the neighbourhood for the i-th pressure extends from the (i-1)-th pressure to the (i+1)-th pressure. At the endpoints it is assumed that the outside bin is of the same pressure range as the first inside bin. For other values of e, the neighbourhood is expanded linearly in each direction. If the "lm" method produces warnings about "prediction from a rank-deficient fit", a larger value of "e" should be used.
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

The "approx" method is best for bottle data, in which the usual task is to interpolate from a coarse sampling grid to a finer one. For CTD data, the "boxcar" method is the more common choice, because the task is normally to sub-sample, and some degree of smoothing is usually desired. (The "lm" method is quite slow, and the results are similar to those of the boxcar method.)

Note that a sort of numerical cabeling effect can result from this procedure, but it can be avoided as follows

```
xd <- ctdDecimate(x)
xd[["sigmaTheta"]] <- swSigmaTheta(xd[["salinity"]],xd[["temperature"]],xd[["pressure"]])
```

Value

An object of ctd-class, with pressures that are as set by the "p" parameter and all other properties modified appropriately.

A note about flags

Data-quality flags contained within the original object are ignored by this function, and the returned value contains no such flags. This is because such flags represent an assessment of the original data, not of quantities derived from those data. This function produces a warning to this effect. The recommended practice is to use handleFlags or some other means to deal with flags before calling the present function.

Author(s)

Dan Kelley

References

R.F. Reiniger and C.K. Ross, 1968. A method of interpolation with application to oceanographic data. *Deep Sea Research*, **15**, 185-193.

See Also

The documentation for ctd-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
library(oce)
data(ctd)
plotProfile(ctd, "salinity", ylim=c(10, 0))
p <- seq(0, 45, 1)
ctd2 <- ctdDecimate(ctd, p=p)
lines(ctd2[["salinity"]], ctd2[["pressure"]], col="blue")
p <- seq(0, 45, 1)
ctd3 <- ctdDecimate(ctd, p=p, method=function(x, y, xout)
                    predict(smooth.spline(x, y, df=30), xout)$y)
lines(ctd3[["salinity"]], ctd3[["pressure"]], col="red")
```

---

ctdFindProfiles *Find Profiles within a Tow-Yow CTD Record*

Description

Examine the pressure record looking for extended periods of either ascent or descent, and return either indices to these events or a vector of CTD records containing the events.

Usage

```
ctdFindProfiles(x, cutoff = 0.5, minLength = 10, minHeight = 0.1 *
  diff(range(x[["pressure"]])), smoother = smooth.spline,
  direction = c("descending", "ascending"), breaks, arr.ind = FALSE,
  distinct, debug = getOption("oceDebug"), ...)
```

Arguments

`x`	A `ctd` object, i.e. one inheriting from `ctd-class`.
`cutoff`	criterion on pressure difference; see "Details".
`minLength`	lower limit on number of points in candidate profiles.
`minHeight`	lower limit on height of candidate profiles.
`smoother`	The smoothing function to use for identifying down/up casts. The default is `smooth.spline`, which performs well for a small number of cycles; see "Examples" for a method that is better for a long tow-yo. The return value from `smoother` must be either a list containing an element named `y` or something that can be coerced to a vector with `as.vector`. To turn smoothing off, so that cycles in pressure are determined by simple first difference, set `smoother` to `NULL`.
`direction`	String indicating the travel direction to be selected.
`breaks`	optional integer vector indicating the indices of last datum in each profile stored within `x`. Thus, the first profile in the return value will contain the `x` data from indices 1 to `breaks[1]`. If `breaks` is given, then all other arguments except `x` are ignored. Using `breaks` is handy in cases where other schemes fail, or when the author has independent knowledge of how the profiles are strung together in `x`; see example 3 for how `breaks` might be used for towyo data.
`arr.ind`	Logical indicating whether the array indices should be returned; the alternative is to return a vector of ctd objects.
`distinct`	An optional string indicating how to identify profiles by unique values. Use `"location"` to find profiles by a change in longitude and latitude, or use the name of any of item in the `data` slot in `x`. In these cases, all the other arguments except `x` are ignored. However, if `distinct` is not supplied, the other arguments are handled as described above.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.
`...`	Optional extra arguments that are passed to the smoothing function, `smoother`.

Details

The method works by examining the pressure record. First, this is smoothed using `smoother()` (see "Arguments"), and then the result is first-differenced using `diff`. Median values of the positive and negative first-difference values are then multiplied by `cutoff`. This establishes criteria for any given point to be in an ascending profile, a descending profile, or a non-profile. Contiguous regions are then found, and those that have fewer than `minLength` points are discarded. Then, those that have pressure ranges less than `minHeight` are discarded.

Caution: this method is not well-suited to all datasets. For example, the default value of `smoother` is `smooth.spline`, and this works well for just a few profiles, but poorly for a tow-yo with a long sequence of profiles; in the latter case, it can be preferable to use simpler smoothers (see "Examples"). Also, depending on the sampling protocol, it is often necessary to pass the resultant profiles through ctdTrim, to remove artifacts such as an equilibration phase, etc. Generally, one is well-advised to use the present function for a quick look at the data, relying on e.g. plotScan to identify profiles visually, for a final product.

Value

If arr.ind=TRUE, a data frame with columns start and end, the indices of the downcasts. Otherwise, a vector of ctd objects. In this second case, the station names are set to a form like "10/3", for the third profile within an original ctd object with station name "10", or to "3", if the original ctd object had no station name defined.

Author(s)

Dan Kelley and Clark Richards

See Also

The documentation for ctd-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
## Not run:
library(oce)
## Example 1.
d <- read.csv("towyow.csv", header=TRUE)
towyow <- as.ctd(d$salinity, d$temperature, d$pressure)

casts <- ctdFindProfiles(towyow)
par(mfrow=c(length(casts), 3))
for (cast in casts) {
  plotProfile(cast, "salinity")
  plotProfile(cast, "temperature")
  plotTS(cast, type='o')
}

## Example 2.
## Using a moving average to smooth pressure, instead of the default
## smooth.spline() method. This avoids a tendency of smooth.spline()
## to smooth out the profiles in a tow-yo with many (dozens or more) cycles.
movingAverage <- function(x, n = 11, ...)
{
   f <- rep(1/n, n)
   stats::filter(x, f, ...)
}
casts <- ctdFindProfiles(towyo, smoother=movingAverage)
```

```
## Example 3: glider data, with profiles separated by >10dbar jump.
breaks <- which(diff(ctd[["pressure"]]) > 10))
profiles <- ctdFindProfiles(ctd, breaks=breaks)

## End(Not run)
```

---

ctdRaw *Seawater CTD Profile, Without Trimming of Extraneous Data*

Description

This is sample CTD profile provided for testing. It includes not just the (useful) portion of the dataset during which the instrument was being lowered, but also data from the upcast and from time spent near the surface. Spikes are also clearly evident in the pressure record. With such real-world wrinkles, this dataset provides a good example of data that need trimming with `ctdTrim`.

Usage

```
data(ctdRaw)
```

Details

This station was sampled by students enrolled in the Dan Kelley's Physical Oceanography class at Dalhousie University. The data were acquired near the centre of the Bedford Basin of the Halifax Harbour, during an October 2003 field trip of Dalhousie University's Oceanography 4120/5120 class. The original `.cnv` data file had temperature in the IPTS-68 scale, but this was converted to the more modern scale using `T90fromT68`.

See Also

A similar dataset (trimmed to the downcast) is available as `data(ctd)`.

Other things related to `ctd` data: `[[,ctd-method`, `[[<-,ctd-method`, `as.ctd`, `cnvName2oceName`, `ctd-class`, `ctdDecimate`, `ctdFindProfiles`, `ctdTrim`, `ctd`, `handleFlags,ctd-method`, `oceNames2whpNames`, `oceUnits2whpUnits`, `plot,ctd-method`, `plotProfile`, `plotScan`, `plotTS`, `read.ctd.itp`, `read.ctd.odf`, `read.ctd.sbe`, `read.ctd.woce.other`, `read.ctd.woce`, `read.ctd`, `subset,ctd-method`, `summary,ctd-method`, `woceNames2oceNames`, `woceUnit2oceUnit`, `write.ctd`

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind` ctdTrim *Trim Beginning and Ending of a CTD cast*

Description

Often in CTD profiling, the goal is to isolate only the downcast, discarding measurements made in the air, in an equilibration phase in which the device is held below the water surface, and then the upcast phase that follows the downcast. This is handled reasonably well by `ctdTrim` with `method="downcast"`, although it is almost always best to use `plotScan` to investigate the data, and then use the `method="index"` or `method="scan"` method based on visual inspection of the data.

Usage

```
ctdTrim(x, method, removeDepthInversions = FALSE, parameters = NULL,
  debug = getOption("oceDebug"))
```

Arguments

`x`	A `ctd` object, i.e. one inheriting from `ctd-class`.
`method`	A string (or a vector of two strings) specifying the trimming method, or a function to be used to determine data indices to keep. If `method` is not provided, `"downcast"` is assumed. See "Details".
`removeDepthInversions`	Logical value indicating whether to remove any levels at which depth is less than, or equal to, a depth above. (This is needed if the object is to be assembled into a section, unless `ctdDecimate` will be used, which will remove the inversions.
`parameters`	A list whose elements depend on the method; see "Details".
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

`ctdTrim` begins by examining the pressure differences between subsequent samples. If these are all of the same value, then the input `ctd` object is returned, unaltered. This handles the case of pressure-binned data. However, if the pressure difference varies, a variety of approaches are taken to trimming the dataset.

- If `method[1]` is `"downcast"` then an attempt is made to keep only data for which the CTD is descending. This is done in stages, with variants based on `method[2]`, if supplied. *Step 1.* The pressure data are despiked with a smooth() filter with method "3R". This removes wild spikes that arise from poor instrument connections, etc. *Step 2.* If no `parameters` are given, then any data with negative pressures are deleted. If there is a parameter named pmin, then that pressure (in decibars) is used instead as the lower limit. This is a commonly-used setup, e.g. ctdTrim(ctd, parameters=list(pmin=1)) removes the top decibar (roughly 1m) from the data. Specifying pmin is a simple way to remove near-surface data, such as a shallow equilibration phase, and if specified will cause ctdTrim to skip step 4 below. *Step 3.* The maximum pressure is determined, and data acquired subsequent to that point are deleted. This removes the upcast and any subsequent data. *Step 4.* If the pmin parameter is not specified, an attempt is made to remove an initial equilibrium phase by a regression of pressure on scan number. There are three variants to this, depending on the value of the second method element. If it is "A" (or not given), the procedure is to call nls to fit a piecewise linear model of pressure as a function of scan, in which pressure is constant for scan less than a critical value, and then linearly varying for with scan. This is meant to handle the common situation in which the CTD is held at roughly constant depth (typically a metre or so) to equilibrate, before it is lowered through the water column. Case "B" is the same, except that the pressure in the surface region is taken to be zero (this does not make much sense, but it might help in some cases). Note that, prior to early 2016, method "B" was called method "C"; the old "B" method was judged useless and was removed.

- If method="upcast", a sort of reverse of "downcast" is used. This was added in late April 2017 and has not been well tested yet.
- If method="sbe", a method similar to that described in the SBE Data Processing manual is used to remove the "soak" period at the beginning of a cast (see Section 6 under subsection "Loop Edit"). The method is based on the soak procedure whereby the instrument sits at a fixed depth for a period of time, after which it is raised toward the surface before beginning the actual downcast. This enables equilibration of the sensors while still permitting reasonably good near-surface data. Parameters for the method can be passed using the parameters argument, which include minSoak (the minimum depth for the soak) and maxSoak the maximum depth of the soak. The method finds the minimum pressure prior to the maxSoak value being passed, each of which occuring after the scan in which the minSoak value was reached. For the method to work, the pre-cast pressure minimum must be less than the minSoak value. The default values of minSoak and maxSoak are 1 and 20 dbar, respectively.
- If method="index" or "scan", then each column of data is subsetted according to the value of parameters. If the latter is a logical vector of length matching data column length, then it is used directly for subsetting. If parameters is a numerical vector with two elements, then the index or scan values that lie between parameters[1] and parameters[2] (inclusive) are used for subsetting. The two-element method is probably the most useful, with the values being determined by visual inspection of the results of plotScan. While this may take a minute or two, the analyst should bear in mind that a deep-water CTD profile might take 6 hours, corresponding to ship-time costs exceeding a week of salary.
- If method="range" then data are selected based on the value of the column named parameters$item. This may be by range or by critical value. By range: select values between parameters$from (the lower limit) and parameters$to (the upper limit) By critical value: select if the named column exceeds the value. For example, ctd2 <- ctdTrim(ctd, "range", parameters=list(item="scan", fr starts at scan number 5 and continues to the end, while ctdTrim(ctd,"range",parameters=list(item="scan",from also starts at scan 5, but extends only to scan 100.
- If method is a function, then it must return a vector of logical values, computed based on two arguments: data (a list), and parameters as supplied to ctdTrim. Both inferWaterDepth and removeInversions are ignored in the function case. See “Examples”.

Value

An object of ctd-class, with data having been trimmed in some way.

Author(s)

Dan Kelley and Clark Richards

References

The Seabird CTD instrument is described at http://www.seabird.com/products/spec_sheets/19plusdata.htm.

Seasoft V2: SBE Data Processing, SeaBird Scientific, 05/26/2016

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
## Not run:
library(oce)
data(ctdRaw)
plot(ctdRaw) # barely recognizable, due to pre- and post-cast junk
plot(ctdTrim(ctdRaw)) # looks like a real profile ...
plot(ctdDecimate(ctdTrim(ctdRaw),method="boxcar")) # ... smoothed
# Demonstrate use of a function. The scan limits were chosen
# by using locator(2) on a graph made by plotScan(ctdRaw).
trimByIndex<-function(data, parameters) {
  parameters[1] < data$scan & data$scan < parameters[2]
}
trimmed <- ctdTrim(ctdRaw, trimByIndex, parameters=c(130, 380))
plot(trimmed)

## End(Not run)
```

---

ctdUpdateHeader *Update a CTD Header [deprecated]*

Description

WARNING: This function will be removed soon; see oce-deprecated.

Usage

```
ctdUpdateHeader(x, debug = FALSE)
```

Arguments

x	A ctd object, i.e. one inheriting from ctd-class.

debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

Update the header of a ctd object, e.g. adjusting nvalues and the span of each column. This is done automatically by ctdTrim, for example.

Value

A new ctd-class object.

Author(s)

Dan Kelley

References

The Seabird CTD instrument is described at http://www.seabird.com/products/spec_sheets/19plusdata.htm.

See Also

Other functions that will be removed soon: addColumn, ctdAddColumn, mapMeridians, mapZones, oce.as.POSIXlt

Examples

```
library(oce)
data(ctd)
ctd[["pressure"]] <- ctd[["pressure"]] + 3
ctdNew <- ctdUpdateHeader(ctd)
```

---

ctimeToSeconds *Interpret a character string as a time interval*

Description

Interpret a character string as a time interval Strings are of the form MM:SS or HH:MM:SS.

Usage

```
ctimeToSeconds(ctime)
```

Arguments

ctime	a character string (see 'Details'.

Value

A numeric value, the number of seconds represented by the string.

Author(s)

Dan Kelley

See Also

See secondsToCtime, the inverse of this.

Other things related to time: julianCenturyAnomaly, julianDay, numberAsHMS, numberAsPOSIXct, secondsToCtime, unabbreviateYear

Examples

```
library(oce)
cat("10      = ", ctimeToSeconds("10"), "s\n", sep="")
cat("01:04   = ", ctimeToSeconds("01:04"), "s\n", sep="")
cat("1:00:00 = ", ctimeToSeconds("1:00:00"), "s\n", sep="")
```

---

curl *Curl of 2D vector field*

Description

Calculate the z component of the curl of an x-y vector field.

Usage

```
curl(u, v, x, y, geographical = FALSE, method = 1)
```

Arguments

u	matrix containing the 'x' component of a vector field
v	matrix containing the 'y' component of a vector field
x	the x values for the matrices, a vector of length equal to the number of rows in u and v.
y	the y values for the matrices, a vector of length equal to the number of cols in u and v.
geographical	logical value indicating whether x and y are longitude and latitude, in which case spherical trigonometry is used.
method	A number indicating the method to be used to calculate the first-difference approximations to the derivatives. See "Details".

Details

The computed component of the curl is defined by $\partial v/\partial x - \partial u/\partial y$ and the estimate is made using first-difference approximations to the derivatives. Two methods are provided, selected by the value of `method`.

- For `method=1`, a centred-difference, 5-point stencil is used in the interior of the domain. For example, $\partial v/\partial x$ is given by the ratio of $v_{i+1,j} - v_{i-1,j}$ to the x extent of the grid cell at index j. (The cell extents depend on the value of `geographical`.) Then, the edges are filled in with nearest-neighbour values. Finally, the corners are filled in with the adjacent value along a diagonal. If `geographical=TRUE`, then `x` and `y` are taken to be longitude and latitude in degrees, and the earth shape is approximated as a sphere with radius 6371km. The resultant `x` and `y` are identical to the provided values, and the resultant `curl` is a matrix with dimension identical to that of `u`.
- For `method=2`, each interior cell in the grid is considered individually, with derivatives calculated at the cell center. For example, $\partial v/\partial x$ is given by the ratio of $0.5*(v_{i+1,j}+v_{i+1,j+1}) - 0.5*(v_{i,j}+v_{i,j+1})$ to the average of the x extent of the grid cell at indices j and $j+1$. (The cell extents depend on the value of `geographical`.) The returned `x` and `y` values are the mid-points of the supplied values. Thus, the returned `x` and `y` are shorter than the supplied values by 1 item, and the returned `curl` matrix dimensions are similarly reduced compared with the dimensions of `u` and `v`.

Value

A list containing vectors `x` and `y`, along with matrix `curl`. See "Details" for the lengths and dimensions, for various values of `method`.

Development status.

This function is under active development as of December 2014 and is unlikely to be stabilized until February 2015.

Author(s)

Dan Kelley and Chantelle Layton

See Also

Other functions relating to vector calculus: grad

Examples

```
library(oce)
## 1. Shear flow with uniform curl.
x <- 1:4
y <- 1:10
u <- outer(x, y, function(x, y) y/2)
v <- outer(x, y, function(x, y) -x/2)
C <- curl(u, v, x, y, FALSE)

## 2. Rankine vortex: constant curl inside circle, zero outside
rankine <- function(x, y)
{
    r <- sqrt(x^2 + y^2)
    theta <- atan2(y, x)
    speed <- ifelse(r < 1, 0.5*r, 0.5/r)
    list(u=-speed*sin(theta), v=speed*cos(theta))
}
x <- seq(-2, 2, length.out=100)
y <- seq(-2, 2, length.out=50)
u <- outer(x, y, function(x, y) rankine(x, y)$u)
v <- outer(x, y, function(x, y) rankine(x, y)$v)
C <- curl(u, v, x, y, FALSE)
## plot results
par(mfrow=c(2, 2))
imagep(x, y, u, zlab="u", asp=1)
imagep(x, y, v, zlab="v", asp=1)
imagep(x, y, C$curl, zlab="curl", asp=1)
hist(C$curl, breaks=100)
```

---

dataLabel *Try to associate data names with units, for use by summary()*

Description

Note that the whole object is not being given as an argument; possibly this will reduce copying and thus storage impact.

Usage

```
dataLabel(names, units)
```

Arguments

names	the names of data within an object
units	the units from metadata

Value

a vector of strings, with blank entries for data with unknown units

Examples

```
library(oce)
data(ctd)
dataLabel(names(ctd@data), ctd@metadata$units)
```

---

decimate *Smooth and Decimate, or Subsample, an Oce Object*

Description

Later on, other methods will be added, and `ctdDecimate` will be retired in favour of this, a more general, function. The filtering is done with the `filter` function of the stats package.

Usage

```
decimate(x, by = 10, to, filter, debug = getOption("oceDebug"))
```

Arguments

`x`	an oce object containing a `data` element.
`by`	an indication of the subsampling. If this is a single number, then it indicates the spacing between elements of `x` that are selected. If it is two numbers (a condition only applicable if `x` is an `echosounder` object, at present), then the first number indicates the time spacing and the second indicates the depth spacing.
`to`	Indices at which to subsample. If given, this over-rides `by`.
`filter`	optional list of numbers representing a digital filter to be applied to each variable in the `data` slot of `x`, before decimation is done. If not supplied, then the decimation is done strictly by sub-sampling.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Value

An object of `class` `"oce"` that has been subsampled appropriately.

Bugs

Only a preliminary version of this function is provided in the present package. It only works for objects of class `echosounder`, for which the decmation is done after applying a running median filter and then a boxcar filter, each of length equal to the corresponding component of `by`.

Author(s)

Dan Kelley

See Also

Filter coefficients may be calculated using `makeFilter`. (Note that `ctdDecimate` will be retired when the present function gains equivalent functionality.)

Examples

```
library(oce)
data(adp)
plot(adp)
adpDec <- decimate(adp,by=2,filter=c(1/4, 1/2, 1/4))
plot(adpDec)
```

---

decodeHeaderNortek *Decode a Nortek Header*

Description

Decode data in a Nortek ADV or ADP header.

Usage

```
decodeHeaderNortek(buf, type = c("aquadoppHR", "aquadoppProfiler", "aquadopp",
  "vector"), debug = getOption("oceDebug"), ...)
```

Arguments

`buf`	a "raw" buffer containing the header
`type`	type of device
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	additional arguments, passed to called routines.

Details

Decodes the header in a binary-format Nortek ADV/ADP file. This function is designed to be used by `read.adp` and `read.adv`, but can be used directly as well. The code is based on information in the Nortek System Integrator Guide (2008) and on postings on the Nortek "knowledge center" discussion board. One might assume that the latter is less authoritative than the former. For example, the inference of cell size follows advice found at `http://www.nortekusa.com/en/knowledge-center/forum/hr-profilers/736804717` (downloaded June 2012)), which contains a typo in an early posting that is corrected later on.

Value

A list containing elements `hardware`, `head`, `user` and `offset`. The easiest way to find the contents of these is to run this function with `debug=3`.

Author(s)

Dan Kelley and Clark Richards

References

1. Information on Nortek profilers (including the System Integrator Guide, which explains the data format byte-by-byte) is available at `http://www.nortekusa.com/`. (One must join the site to see the manuals.)

2. The Nortek Knowledge Center `http://www.nortekusa.com/en/knowledge-center` may be of help if problems arise in dealing with data from Nortek instruments.

See Also

Most users should employ the functions `read.adp` and `read.adv` instead of this one.

`decodeTime` *Oce Version of as.POSIXct*

Description

Oce Version of as.POSIXct

Usage

```
decodeTime(time, timeFormats, tz = "UTC")
```

Arguments

`time`	Character string with an indication of the time.
`timeFormats`	Optional vector of time formats to use, as for `as.POSIXct`.
`tz`	Time zone.

Details

Each format in `timeFormats` is used in turn as the `format` argument to `as.POSIXct`, and the first that produces a non-`NA` result is used. If `timeFormats` is missing, the following formats are tried, in the stated order:

- `"%b %d %Y %H:%M:%S"` (e.g. `"Jul 1 2013 01:02:03"`)
- `"%b %d %Y"` (e.g. `"Jul 1 2013"`)
- `"%B %d %Y %H:%M:%S"` (e.g. `"July 1 2013 01:02:03"`)

- "%B %d %Y" (e.g. "July 1 2013")
- "%d %b %Y %H:%M:%S" (e.g. "1 Jul 2013 01:02:03")
- "%d %b %Y" (e.g. "1 Jul 2013")
- "%d %B %Y %H:%M:%S" (e.g. "1 July 2013 01:02:03")
- "%d %B %Y" (e.g. "1 July 2013")
- "%Y-%m-%d %H:%M:%S" (e.g. "2013-07-01 01:02:03")
- "%Y-%m-%d" (e.g. "2013-07-01")
- "%Y-%b-%d %H:%M:%S" (e.g. "2013-July-01 01:02:03")
- "%Y-%b-%d" (e.g. "2013-Jul-01")
- "%Y-%B-%d %H:%M:%S" (e.g. "2013-July-01 01:02:03")
- "%Y-%B-%d" (e.g. "2013-July-01")
- "%d-%b-%Y %H:%M:%S" (e.g. "01-Jul-2013 01:02:03")
- "%d-%b-%Y" (e.g. "01-Jul-2013")
- "%d-%B-%Y %H:%M:%S" (e.g. "01-July-2013 01:02:03")
- "%d-%B-%Y" (e.g. "01-July-2013")
- "%Y/%b/%d %H:%M:%S" (e.g. "2013/Jul/01 01:02:03")
- "%Y/%b/%d" (e.g. "2013/Jul/01")
- "%Y/%B/%d %H:%M:%S" (e.g. "2013/July/01 01:02:03")
- "%Y/%B/%d" (e.g. "2013/July/01")
- "%Y/%m/%d %H:%M:%S" (e.g. "2013/07/01 01:02:03")
- "%Y/%m/%d" (e.g. "2013/07/01")

Value

A time as returned by as.POSIXct.

Author(s)

Dan Kelley

See Also

Other functions relating to time: GMTOffsetFromTz

Examples

```
decodeTime("July 1 2013 01:02:03")
decodeTime("Jul 1 2013 01:02:03")
decodeTime("1 July 2013 01:02:03")
decodeTime("1 Jul 2013 01:02:03")
decodeTime("2013-07-01 01:02:03")
decodeTime("2013/07/01 01:02:03")
decodeTime("2013/07/01")
```

despike *Remove spikes from a time series*

Description

The method identifies spikes with respect to a "reference" time-series, and replaces these spikes with the reference value, or with NA according to the value of action; see “Details”.

Usage

```
despike(x, reference = c("median", "smooth", "trim"), n = 4, k = 7,
  min = NA, max = NA, replace = c("reference", "NA"), skip)
```

Arguments

x	a vector of (time-series) values, a list of vectors, a data frame, or an object that inherits from class oce.
reference	indication of the type of reference time series to be used in the detection of spikes; see ‘Details’.
n	an indication of the limit to differences between x and the reference time series, used for reference="median" or reference="smooth"; see ‘Details.’
k	length of running median used with reference="median", and ignored for other values of reference.
min	minimum non-spike value of x, used with reference="trim".
max	maximum non-spike value of x, used with reference="trim".
replace	an indication of what to do with spike values, with "reference" indicating to replace them with the reference time series, and "NA" indicating to replace them with NA.
skip	optional vector naming columns to be skipped. This is ignored if x is a simple vector. Any items named in skip will be passed through to the return value without modification. In some cases, despike will set up reasonable defaults for skip, e.g. for a ctd object, skip will be set to c("time", "scan","pressure") if it is not supplied as an argument.

Details

Three modes of operation are permitted, depending on the value of reference.

- For reference="median", the first step is to linearly interpolate across any gaps (spots where x==NA), using approx with rule=2. The second step is to pass this through runmed to get a running median spanning k elements. The result of these two steps is the "reference" time-series. Then, the standard deviation of the difference between x and the reference is calculated. Any x values that differ from the reference by more than n times this standard deviation are considered to be spikes. If replace="reference", the spike values are replaced with the reference, and the resultant time series is returned. If replace="NA", the spikes are replaced with NA, and that result is returned.

- For reference="smooth", the processing is the same as for "median", except that smooth is used to calculate the reference time series.
- For reference="trim", the reference time series is constructed by linear interpolation across any regions in which x<min or x>max. (Again, this is done with approx with rule=2.) In this case, the value of n is ignored, and the return value is the same as x, except that spikes are replaced with the reference series (if replace="reference" or with NA, if replace="NA".

Value

A new vector in which spikes are replaced as described above.

Author(s)

Dan Kelley

Examples

```
n <- 50
x <- 1:n
y <- rnorm(n=n)
y[n/2] <- 10                    # 10 standard deviations
plot(x, y, type='l')
lines(x, despike(y), col='red')
lines(x, despike(y, reference="smooth"), col='darkgreen')
lines(x, despike(y, reference="trim", min=-3, max=3), col='blue')
legend("topright", lwd=1, col=c("black", "red", "darkgreen", "blue"),
       legend=c("raw", "median", "smooth", "trim"))

# add a spike to a CTD object
data(ctd)
plot(ctd)
T <- ctd[["temperature"]]
T[10] <- T[10] + 10
ctd[["temperature"]] <- T
CTD <- despike(ctd)
plot(CTD)
```

---

detrend *Detrend a set of observations*

Description

Detrends y by subtracting a linear trend in x, to create a vector that is zero for its first and last finite value. If the second parameter (y) is missing, then x is taken to be y, and a new x is constructed with seq_along. Any NA values are left as-is.

Usage

```
detrend(x, y)
```

Arguments

x	a vector of numerical values. If y is not given, then x is taken for y.
y	an optional vector

Details

A common application is to bring the end points of a time series down to zero, prior to applying a digital filter. (See examples.)

Value

A list containing Y, the detrended version of y, and the intercept a and slope b of the linear function of x that is subtracted from y to yield Y.

Author(s)

Dan Kelley

Examples

```
x <- seq(0, 0.9 * pi, length.out=50)
y <- sin(x)
y[1] <- NA
y[10] <- NA
plot(x, y, ylim=c(0, 1))
d <- detrend(x, y)
points(x, d$Y, pch=20)
abline(d$a, d$b, col='blue')
abline(h=0)
points(x, d$Y + d$a + d$b * x, col='blue', pch='+')
```

---

download.amsr — *Download and Cache an amsr File*

Description

If the file is already present in destdir, then it is not downloaded again. The default destdir is the present directory, but it probably makes more sense to use something like "~/data/amsr" to make it easy for scripts in other directories to use the cached data. The file is downloaded with download.file.

Usage

```
download.amsr(year, month, day, destdir = ".",
  server = "http://data.remss.com/amsr2/bmaps_v08")
```

Arguments

`year, month, day`
: Numerical values of the year, month, and day of the desired dataset. Note that one file is archived per day, so these three values uniquely identify a dataset. If `day` and `month` are not provided but `day` is, then the time is provided in a relative sense, based on the present date, with `day` indicating the number of days in the past. Owing to issues with timezones and the time when the data are uploaded to the server, `day=3` may yield the most recent available data. For this reason, there is a third option, which is to leave `day` unspecified, which works as though `day=3` had been given.

`destdir`
: A string naming the directory in which to cache the downloaded file. The default is to store in the present directory, but many users find it more helpful to use something like `"~/data/amsr"` for this, to collect all downloaded amsr files in one place.

`server`
: A string naming the server from which data are to be acquired. See “History”.

Value

A character value indicating the filename of the result; if there is a problem of any kind, the result will be the empty string.

History

Until 25 March 2017, the default server was `"ftp.ssmi.com/amsr2/bmaps_v07.2"`, but this was changed when the author discovered that this FTP site had been changed to require users to create accounts to register for downloads. The default was changed to `"http://data.remss.com/amsr2/bmaps_v07.2"` on the named date. This site was found by a web search, but it seems to provide proper data. It is assumed that users will do some checking on the best source.

On 23 January 2018, it was noticed that the server-url naming convention had changed, e.g. `http://data.remss.com/amsr2` becoming `http://data.remss.com/amsr2/bmaps_v08/y2017/m01/f34_20170114v8.gz`

References

http://images.remss.com/amsr/amsr2_data_daily.html provides daily images going back to 2012. Three-day, monthly, and monthly composites are also provided on that site.

See Also

Other functions that download files: `download.coastline`, `download.met`, `download.topo`

Other things related to amsr data: `[[<-,amsr-method`, `amsr-class`, `composite,amsr-method`, `plot,amsr-method`, `read.amsr`, `subset,amsr-method`, `summary,amsr-method`

Examples

```
## Not run:
## The download takes several seconds.
f <- download.amsr(2017, 1, 14) # Jan 14, 2017
d <- read.amsr(f)
```

```
plot(d)
mtext(d[["filename"]], side=3, line=0, adj=0)

## End(Not run)
```

---

download.coastline *Download a coastline File*

Description

Constructs a query to the NaturalEarth server [1] to download coastline data (or lake data, river data, etc) in any of three resolutions.

Usage

```
download.coastline(resolution, item = "coastline", destdir = ".", destfile,
  server = "naturalearth", debug = getOption("oceDebug"))
```

Arguments

`resolution`	A character value specifying the desired resolution. The permitted choices are `"10m"` (for 1:10M resolution, the most detailed), `"50m"` (for 1:50M resolution) and `"110m"` (for 1:110M resolution). If `resolution` is not supplied, `"50m"` will be used.
`item`	A character value indicating the quantity to be downloaded. This is normally one of `"coastline"`, `"land"`, `"ocean"`, `"rivers_lakes_centerlines"`, or `"lakes"`, but the NaturalEarth server has other types, and advanced users can discover their names by inspecting the URLs of links on the NaturalEarth site, and use them for `item`. If `item` is not supplied, it defaults to `"coastline"`.
`destdir`	Optional string indicating the directory in which to store downloaded files. If not supplied, `"."` is used, i.e. the data file is stored in the present working directory.
`destfile`	Optional string indicating the name of the file. If not supplied, the file name is constructed from the other parameters of the function call, so subsequent calls with the same parameters will yield the same result, thus providing the key to the caching scheme.
`server`	A character value specifying the server that is to supply the data. At the moment, the only permitted value is `"naturalearth"`, which is the default if `server` is not supplied.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Value

A character value indicating the filename of the result; if there is a problem of any kind, the result will be the empty string.

References

1. The NaturalEarth server is at http://www.naturalearthdata.com

See Also

The work is done with download.file.

Other functions that download files: download.amsr, download.met, download.topo

Other things related to coastline data: [[,coastline-method, [[<-,coastline-method, as.coastline, coastline-class, coastlineBest, coastlineCut, coastlineWorld, plot,coastline-method, read.coastline.openstreetmap, read.coastline.shapefile, subset,coastline-method, summary,coastline-method

Examples

```
## Not run:
library(oce)
# User must create directory ~/data/coastline first.
# As of September 2016, the downloaded file, named
# "ne_50m_coastline.zip", occupies 443K bytes.
filename <- download.coastline(destdir="~/data/coastline")
coastline <- read.coastline(filename)
plot(coastline)

## End(Not run)
```

---

download.met *Download and Cache a met File*

Description

Data are downloaded from http://climate.weather.gc.ca and cached locally.

Usage

```
download.met(id, year, month, deltat, destdir = ".", destfile,
  debug = getOption("oceDebug"))
```

Arguments

- `id` — A number giving the "Station ID" of the station of interest. If not provided, `id` defaults to 6358, for Halifax International Airport. See "Details".
- `year` — A number giving the year of interest. Ignored unless `deltat` is `"hour"`. If `year` is not given, it defaults to the present year.
- `month` — A number giving the month of interest. Ignored unless `deltat` is `"hour"`. If `month` is not given, it defaults to the present month.
- `deltat` — Optional character string indicating the time step of the desired dataset. This may be `"hour"` or `"month"`. If `deltat` is not given, it defaults to `"hour"`.
- `destdir` — Optional string indicating the directory in which to store downloaded files. If not supplied, `"."` is used, i.e. the data file is stored in the present working directory.
- `destfile` — Optional string indicating the name of the file. If not supplied, the file name is constructed from the other parameters of the function call, so subsequent calls with the same parameters will yield the same result, thus providing the key to the caching scheme.
- `debug` — an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

The data are downloaded with `download.file` pointed to the Environment Canada website [1] using queries that had to be devised by reverse-engineering, since the agency does not provide documentation about how to construct queries. Caution: the query format changes from time to time, so `download.met` may work one day, and fail the next.

The constructed query contains Station ID, as provided in the `id` argument. Note that this seems to be a creation of Environment Canada, alone; it is distinct from the more standard "Climate ID" and "WMO ID". To make things more difficult, Environment Canada states that the Station ID is subject to change over time. (Whether this applies to existing data is unclear.)

Given these difficulties with Station ID, users are advised to consult the Environment Canada website [1] before downloading any data, and to check it from time to time during the course of a research project, to see if the Station ID has changed. Another approach would be to use Gavin Simpson's `canadaHCD` package [2] to look up Station IDs. This package maintains a copy of the Environment Canada listing of stations, and its `find_station` function provides an easy way to determine Station IDs. After that, its `hcd_hourly` function (and related functions) make it easy to read data. These data can then be converted to the `met` class with `as.met`, although doing so leaves many important metadata blank.

Value

String indicating the full pathname to the downloaded file.

Author(s)

Dan Kelley

References

1. Environment Canada website for Historical Climate Data http://climate.weather.gc.ca/index_e.html

2. Gavin Simpon's canadaHCD package on GitHub https://github.com/gavinsimpson/canadaHCD

See Also

The work is done with download.file.

Other functions that download files: download.amsr, download.coastline, download.topo

Other things related to met data: [[,met-method, [[<-,met-method, as.met, met-class, met, plot,met-method, read.met, subset,met-method, summary,met-method

Examples

```
## Not run:
library(oce)
## Download data for Halifax International Airport, in September
## of 2003. (This dataset is used for data(met) provided with oce.)
metFile <- download.met(6358, 2003, 9, destdir=".")
met <- read.met(metFile)

## End(Not run)
```

---

download.topo *Download and Cache a topo File*

Description

Data are downloaded (from 'https://maps.ngdc.noaa.gov/viewers/wcs-client/', by default) and a string containing the full path to the downloaded file is returned. Typically, this return value is used with read.topo to read the data. Subsequent calls to download.topo with identical parameters will simply return the name of the cached file, assuming the user has not deleted it in the meantime. For convenience, if destfile is not given, then download.topo will construct a filename from the other arguments, rounding longitude and latitude limits to 0.01 degrees.

Usage

```
download.topo(west, east, south, north, resolution, destdir, destfile, format,
  server, debug = getOption("oceDebug"))
```

Arguments

west, east	Longitudes of the western and eastern sides of the box.
south, north	Latitudes of the southern and northern sides of the box.
resolution	Optional grid spacing, in minutes. If not supplied, a default value of 4 (corresponding to 7.4km, or 4 nautical miles) is used. Note that (as of August 2016) the original data are on a 1-minute grid, which limits the possibilities for resolution.
destdir	Optional string indicating the directory in which to store downloaded files. If not supplied, "." is used, i.e. the data file is stored in the present working directory.
destfile	Optional string indicating the name of the file. If not supplied, the file name is constructed from the other parameters of the function call, so subsequent calls with the same parameters will yield the same result, thus providing the key to the caching scheme.
format	Optional string indicating the type of file to download. If not supplied, this defaults to "gmt". See "Details".
server	Optional string indicating the server from which to get the data. If not supplied, the default '"https://gis.ngdc.noaa.gov/cgi-bin/public/wcs/etopo1.xyz"' will be used.
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

The data are downloaded with download.file, using a URL devised from reverse engineering web-based queries constructed by the default server used here. Note that the data source is "etopo1", which is a 1 arc-second file [1,2].

Three values are permitted for format, each named after the targets of menu items on the NOAA website (as of August 2016): (1) "aaigrid" (for the menu item "ArcGIS ASCII Grid"), which yields a text file, (2) "netcdf" (the default, for the menu item named "NetCDF"), which yields a NetCDF file and (3) "gmt" (for the menu item named "GMT NetCDF"), which yields a NetCDF file in another format. All of these file formats are recognized by read.topo. (The NOAA server has more options, and if read.topo is extended to handle them, they will also be added here.)

Value

String indicating the full pathname to the downloaded file.

Webserver history

All versions of download.topo to date have used a NOAA server as the data source, but the URL has not been static. A list of the servers that have been used is provided below, in hopes that it can help users to make guesses for server, should download.topo fail because of a fail to download the data. Another hint is to look at the source code for `getNOAA.bathy` in the **marmap** package, which is also forced to track the moving target that is NOAA.

- August 2016. '`http://maps.ngdc.noaa.gov/mapviewer-support/wcs-proxy/wcs.groovy`'
- December 2016. '`http://mapserver.ngdc.noaa.gov/cgi-bin/public/wcs/etopo1.xyz`'
- June-September 2017. '`https://gis.ngdc.noaa.gov/cgi-bin/public/wcs/etopo1.xyz`'

Author(s)

Dan Kelley

References

1. '`https://www.ngdc.noaa.gov/mgg/global/global.html`'

2. Amante, C. and B.W. Eakins, 2009. ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis. NOAA Technical Memorandum NESDIS NGDC-24. National Geophysical Data Center, NOAA. doi:10.7289/V5C8276M [access date: Aug 30, 2017].

See Also

The work is done with `download.file`.

Other functions that download files: `download.amsr`, `download.coastline`, `download.met`

Other things related to topo data: `[[,topo-method`, `[[<-,topo-method`, `as.topo`, `plot,topo-method`, `read.topo`, `subset,topo-method`, `summary,topo-method`, `topo-class`, `topoInterpolate`, `topoWorld`

Examples

```
## Not run:
library(oce)
topoFile <- download.topo(west=-64, east=-60, south=43, north=46,
                          resolution=1, destdir="~/data/topo")
topo <- read.topo(topoFile)
imagep(topo, zlim=c(-400, 400), drawTriangles=TRUE)
data(coastlineWorldFine, package="ocedata")
lines(coastlineWorldFine[["longitude"]], coastlineWorldFine[["latitude"]])

## End(Not run)
```

`drawDirectionField` *Draw a Direction Field*

Description

The direction field is indicated variously, depending on the value of `type`:

- For `type=1`, each indicator is drawn with a symbol, according to the value of `pch` (either supplied globally, or as an element of the `...` list) and of size `cex`, and colour `col`. Then, a line segment is drawn for each, and for this `lwd` and `col` may be set globally or in the `...` list.
- For `type=2`, the points are not drawn, but arrows are drawn instead of the line segments. Again, `lwd` and `col` control the type of the line.

Usage

```
drawDirectionField(x, y, u, v, scalex, scaley, skip, length = 0.05,
  add = FALSE, type = 1, col = par("fg"), pch = 1, cex = par("cex"),
  lwd = par("lwd"), lty = par("lty"), xlab = "", ylab = "",
  debug = getOption("oceDebug"), ...)
```

Arguments

`x, y`	coordinates at which velocities are specified. The length of `x` and `y` depends on the form of `u` and `v` (vectors or matrices).
`u, v`	velocity components in the `x` and `y` directions. Can be either vectors with the same length as `x, y`, or matrices, of dimension `length(x)` by `length(y)`.
`scalex, scaley`	scale to be used for the velocity arrows. Exactly one of these must be specified. Arrows that have `u^2+v^2=1` will have length `scalex` along the x axis, or `scaley` along the y axis, according to which argument is given.
`skip`	either an integer, or a two-element vector indicating the number of points to skip when plotting arrows (for the matrix `u, v` case). If a single value, the same `skip` is applied to both the `x` and `y` directions. If a two-element vector, specifies different values for the `x` and `y` directions.
`length`	indication of **width** of arrowheads. The somewhat confusing name of this argument is a consequence of the fact that it is passed to `arrows` for drawing arrows. Note that the present default is smaller than the default used by `arrows`.
`add`	if `TRUE`, the arrows are added to an existing plot; otherwise, a new plot is started by calling `plot` with `x`, `y` and `type="n"`. In other words, the plot will be very basic. In most cases, the user will probably want to draw a diagram first, and add the direction field later.
`type`	indication of the style of arrow-like indication of the direction.
`col`	colour of line segments or arrows
`pch, cex`	plot character and expansion factor, used for `type=1`
`lwd, lty`	line width and type, used for `type=2`
`xlab, ylab`	`x` and `y` axis labels
`debug`	debugging value; set to a positive integer to get debugging information.
`...`	other arguments to be passed to plotting functions (e.g. axis labels, etc).

Value

None.

Author(s)

Dan Kelley and Clark Richards

Examples

```
library(oce)
plot(c(-1.5, 1.5), c(-1.5, 1.5), xlab="", ylab="", type='n')
drawDirectionField(x=rep(0, 2), y=rep(0, 2), u=c(1, 1), v=c(1, -1), scalex=0.5, add=TRUE)
plot(c(-1.5, 1.5), c(-1.5, 1.5), xlab="", ylab="", type='n')
drawDirectionField(x=rep(0, 2), y=rep(0, 2), u=c(1, 1), v=c(1, -1), scalex=0.5, add=TRUE,
                   type=2)

## 2D example
x <- seq(-2, 2, 0.1)
y <- x
xx <- expand.grid(x, y)[,1]
yy <- expand.grid(x, y)[,2]
z <- matrix(xx*exp(-xx^2 -yy^2), nrow=length(x))
gz <- grad(z, x, y)
drawDirectionField(x, y, gz$gx, gz$gy, scalex=0.5, type=2, len=0.02)
oceContour(x, y, z, add=TRUE)
```

---

drawIsopycnals *Add Isopycnal Curves to TS Plot*

Description

Adds isopycnal lines to an existing temperature-salinity plot. This is called by `plotTS`, and may be called by the user also, e.g. if an image plot is used to show TS data density.

Usage

```
drawIsopycnals(nlevels = 6, levels, rotate = TRUE, rho1000 = FALSE,
  digits = 2, eos = getOption("oceEOS", default = "gsw"), cex = 0.75 *
  par("cex"), col = "darkgray", lwd = par("lwd"), lty = par("lty"))
```

Arguments

`nlevels`	suggested number of density levels (i.e. isopycnal curves); ignored if `levels` is supplied.
`levels`	optional density levels to draw.
`rotate`	boolean, set to TRUE to write all density labels horizontally.
`rho1000`	boolean, set to TRUE to write isopycnal labels as e.g. 1024 instead of 24.

`digits`	number of decimal digits to use in label (supplied to `round`).
`eos`	equation of state to be used, either `"unesco"` or `"gsw"`.
`cex`	size for labels.
`col`	colour for lines and labels.
`lwd`	line width for isopcynal curves
`lty`	line type for isopcynal curves

Value

None.

Author(s)

Dan Kelley

See Also

`plotTS`, which calls this.

`drawPalette` *Draw a palette, leaving margins suitable for accompanying plot*

Description

Draw a palette, leaving margins suitable for accompanying plot.

Usage

```
drawPalette(zlim, zlab = "", breaks, col, colormap, mai,
  cex.axis = par("cex.axis"), pos = 4, labels = NULL, at = NULL, levels,
  drawContours = FALSE, plot = TRUE, fullpage = FALSE,
  drawTriangles = FALSE, axisPalette, tformat,
  debug = getOption("oceDebug"), ...)
```

Arguments

`zlim`	two-element vector containing the lower and upper limits of z. This may also be a vector of any length exceeding 1, in which case its range is used.
`zlab`	label for the palette scale.
`breaks`	the z values for breaks in the colour scheme.
`col`	either a vector of colours corresponding to the breaks, of length 1 less than the number of breaks, or a function specifying colours, e.g. `oce.colorsJet` for a rainbow.
`colormap`	a colour map as created by `colormap`. If provided, this takes precedence over `breaks` and `col`.

mai	margins for palette, as defined in the usual way; see par. If not given, reasonable values are inferred from the existence of a non-blank zlab.
cex.axis	character-expansion value for text labels
pos	an integer indicating the location of the palette within the plotting area, 1 for near the bottom, 2 for near the left-hand side, 3 for near the top side, and 4 (the default) for near the right-hand side.
labels	optional vector of labels for ticks on palette axis (must correspond with at)
at	optional vector of positions for the labels
levels	optional contour levels, in preference to breaks values, to be added to the image if drawContours is TRUE.
drawContours	logical value indicating whether to draw contours on the palette, at the colour breaks.
plot	logical value indicating whether to plot the palette, the default, or whether to just alter the margins to make space for where the palette would have gone. The latter case may be useful in lining up plots, as in example 1 of "Examples".
fullpage	logical value indicating whether to draw the palette filling the whole plot width (apart from mai, of course). This can be helpful if the palette panel is to be created with layout, as illustrated in the "Examples".
drawTriangles	logical value indicating whether to draw triangles on the top and bottom of the palette. If a single value is provide, it applies to both ends of the palette. If a pair is provided, the first refers to the lower range of the palette, and the second to the upper range.
axisPalette	optional replacement function for axis(), e.g. for exponential notation on large or small values.
tformat	optional format for axis labels, if the variable is a time type (ignored otherwise).
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Details

In the normal use, drawPalette draws an image palette near the right-hand side of the plotting device, and then adjusts the global margin settings in such a way as to cause the next plot to appear (with much larger width) to the left of the palette. The function can also be used, if zlim is not provided, to adjust the margin without drawing anything; this is useful in lining up the x axes of a stack of plots, some some of which will have palettes and others not.

The plot positioning is done entirely with margins, not with par(mfrow) or other R schemes for multi-panel plots. This means that the user is free to use those schemes without worrying about nesting or conflicts.

Value

None.

Use with multi-panel plots

An important consequence of the margin adjustment is that multi-panel plots require that the initial margin be stored prior to the first call to `drawPalette`, and reset after each palette-plot pair. This method is illustrated in “Examples”.

Author(s)

Dan Kelley, with help from Clark Richards

See Also

This is used by `imagep`.

Examples

```
library(oce)
par(mgp=getOption("oceMgp"))

## 1. A three-panel plot
par(mfrow=c(3, 1), mar=c(3, 3, 1, 1))
omar <- par('mar')                 # save initial margin

## 1a. top panel: simple case
drawPalette(zlim=c(0, 1), col=oce.colorsJet(10))
plot(1:10, 1:10, col=oce.colorsJet(10)[1:10],pch=20,cex=3,xlab='x', ylab='y')
par(mar=omar)                      # reset margin

## 1b. middle panel: colormap
cm <- colormap(name="gmt_globe")
drawPalette(colormap=cm)
icol <- seq_along(cm$col)
plot(icol, cm$breaks[icol], pch=20, cex=2, col=cm$col,
     xlab="Palette index", ylab="Palette breaks")
par(mar=omar)                      # reset margin

## 1c. bottom panel: space for palette (to line up graphs)
drawPalette(plot=FALSE)
plot(1:10, 1:10, col=oce.colorsJet(10)[1:10],pch=20,cex=3,xlab='x', ylab='y')
par(mar=omar)                      # reset margin

# 2. Use layout to mimic the action of imagep(), with the width
# of the palette region being 14 percent of figure width.
d <- 0.14
layout(matrix(1:2,nrow=1), widths=c(1-d,d))
image(volcano, col=oce.colorsJet(100), zlim=c(90, 200))
contour(volcano, add=TRUE)
drawPalette(c(90, 200), fullpage=TRUE, col=oce.colorsJet)
```

`echosounder` *Echosounder Dataset*

Description

This is degraded subsample of measurements that were made with a Biosonics scientific echosounder, as part of the St Lawrence Internal Wave Experiment (SLEIWEX).

Author(s)

Dan Kelley

Source

This file came from the SLEIWEX-2008 experiment, and was decimated using `decimate` with `by=c()`.

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to `echosounder` data: `[[,echosounder-method`, `[[<-,echosounder-method`, `as.echosounder`, `echosounder-class`, `findBottom`, `plot,echosounder-method`, `read.echosounder`, `subset,echosounder-method`, `summary,echosounder-method`

`echosounder-class` *Class to Store Echosounder Data*

Description

Class to store echosounder data.

Details

The `data` slot is a list containing

- An infrequently updated record of the intrument position, in `timeSlow`, `longitudeSlow` and `latitudeSlow`. These are used in plotting maps with `plot,echosounder-method`.
- An interpolated record of the instrument position, in `time`, `longitude`, and `latitude`. Linear interpolation is used to infer the longitude and latitude from the variables listed above.
- `depth`, vector of depths of echo samples (measured positive downwards in the water column). This is calculated from the inter-sample time interval and the sound speed provided as the `soundSpeed` argument to `read.echosounder`, so altering the value of the latter will alter the echosounder plots provided by `plot,echosounder-method`.

- The echosounder signal amplitude a, a matrix whose number of rows matches the length of time, etc., and number of columns equal to the length of depth. Thus, for example, a[100,] represents the depth-dependent amplitude at the time of the 100th ping.
- A matrix named b exists for dual-beam and split-beam cases. For dual-beam data, this is the wide-beam data, whereas a is the narrow-beam data. For split-beam data, this is the x-angle data.
- A matrix named c exists for split-beam data, containing the y-angle data.
- In addition to these matrices, ad-hoc calculated matrices named Sv and TS may be accessed as explained in the next section.

Methods

Accessing values. Data may be accessed as e.g. echosounder[["time"]], echosounder[["depth"]], echosounder[["a"]], etc. Items in metadata must be specified by full name, but those in data may be abbreviated, so long as the abbreviation is unique. In addition to the actual data, some derived fields are also available: echosounder[["distance"]] calls geodDist to compute calculate distance along the ship track, echosounder[["Sv"]] returns a matrix of backscatter strength in DB, and echosounder[["TS"]] returns a matrix of target strength in dB.

Assigning values. Everything that may be accessed may also be assigned, e.g. echosounder[["time"]] <- 3600 + echosou adds an hour to time.

Author(s)

Dan Kelley

Statistical summaries are provided by summary,echosounder-method, while show displays an overview. The findBottom function infers the ocean bottom from tracing the strongest reflector from ping to ping.

Echosounder objects may be plotted with plot,echosounder-method.

The contents of echosounder objects may be altered with subset,echosounder-method, or with the [[]] scheme discussed in the previous section; skilled users may also manipulate the contents directly, but this is not recommended because it is brittle to changes in the data structure.

See Also

Other classes provided by oce: adp-class, adv-class, argo-class, bremen-class, cm-class, coastline-class, ctd-class, lisst-class, lobo-class, met-class, oce-class, odf-class, rsk-class, sealevel-class, section-class, topo-class, windrose-class

Other things related to echosounder data: [[,echosounder-method, [[<-,echosounder-method, as.echosounder, echosounder, findBottom, plot,echosounder-method, read.echosounder, subset,echosounder-method, summary,echosounder-method eclipticalToEquatorial

Convert ecliptical to equatorial coordinate

Description

Convert from ecliptical to equatorial coordinates, using equations 8.3 and 8.4 of [1], or, equivalently, equations 12.3 and 12.4 of [2].

Usage

```
eclipticalToEquatorial(lambda, beta, epsilon)
```

Arguments

`lambda`	longitude, in degrees, or a data frame containing `lambda`, `beta`, and `epsilon`, in which case the next to arguments are ignored.
`beta`	geocentric latitude, in degrees
`epsilon`	obliquity of the ecliptic, in degrees

Value

A data frame containing columns `rightAscension` and `declination` both in degrees.

Author(s)

Dan Kelley, based on formulae in [1] and [2].

References

1. Meeus, Jean, 1982. Astronomical formulae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages.

2. Meeus, Jean, 1991. Astronomical algorithms. Willmann-Bell, Richmond VA, USA. 429 pages.
The code is based on [1]; see help(moonAngle,"oce") for comments on the differences in formulae found in [2]. Indeed, [2] is only cited here in case readers want to check the ideas of the formulae; DK has found that [2] is available to him via his university library inter-library loan system, whereas he owns a copy of [1].

See Also

Other things related to astronomy: `equatorialToLocalHorizontal`, `julianCenturyAnomaly`, `julianDay`, `moonAngle`, `siderealTime`, `sunAngle`

enuToOther — *Rotate acoustic-Doppler data to a new coordinate system*

Description

Rotate acoustic-Doppler data to a new coordinate system

Usage

```
enuToOther(x, ...)
```

Arguments

`x`	an adp or adv object, i.e. one inheriting from `adp-class` or `adv-class`.
`...`	extra arguments that are passed on to `enuToOtherAdp` or `enuToOtherAdv`.

Value

An object of the same type as `x`, but with velocities in the rotated coordinate system

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOtherAdv`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnuAdv`, `xyzToEnu`

enuToOtherAdp — *Convert ADP ENU to Rotated Coordinate*

Description

Convert ADP velocity components from an enu-based coordinate system to another system, perhaps to align axes with the coastline.

Usage

```
enuToOtherAdp(x, heading = 0, pitch = 0, roll = 0)
```

Arguments

x	An adp object, i.e. one inheriting from adp-class.
heading	number or vector of numbers, giving the angle, in degrees, to be added to the heading. See "Details".
pitch	as heading but for pitch.
roll	as heading but for roll.

Details

The supplied angles specify rotations to be made around the axes for which heading, pitch, and roll are defined. For example, an eastward current will point southeast if heading=45 is used.

The returned value has heading, pitch, and roll matching those of x, so these angles retain their meaning as the instrument orientation.

NOTE: this function works similarly to xyzToEnuAdp, except that in the present function, it makes no difference whether the instrument points up or down, etc.

Value

An object with data$v[,1:3,] altered appropriately, and metadata$oce.coordinate changed from enu to other.

Author(s)

Dan Kelley

References

RD Instruments, 1998. *ADP Coordinate Transformation, formulas and calculations.* P/N 951-6079-00 (July 1998)

See Also

See read.adp for other functions that relate to objects of class "adp".

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

Examples

```
library(oce)
data(adp)
o <- enuToOtherAdp(adp, heading=-31.5)
plot(o, which=1:3)
```

enuToOtherAdv *Convert ENU to Other Coordinate*

Description

Convert ADV velocity components from an enu-based coordinate system to another system, perhaps to align axes with the coastline.

Usage

```
enuToOtherAdv(x, heading = 0, pitch = 0, roll = 0,
  debug = getOption("oceDebug"))
```

Arguments

`x`	An adv object, i.e. one inheriting from `adv-class`.
`heading`	number or vector of numbers, giving the angle, in degrees, to be added to the heading. If this has length less than the number of velocity sampling times, then it will be extended using `rep`.
`pitch`	as `heading` but for pitch.
`roll`	as `heading` but for roll.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

The supplied angles specify rotations to be made around the axes for which heading, pitch, and roll are defined. For example, an eastward current will point southeast if `heading=45` is used.

The returned value has heading, pitch, and roll matching those of `x`, so these angles retain their meaning as the instrument orientation.

NOTE: this function works similarly to `xyzToEnuAdv`, except that in the present function, it makes no difference whether the instrument points up or down, etc.

Author(s)

Dan Kelley

See Also

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOther`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnuAdv`, `xyzToEnu` `equatorialToLocalHorizontal` *Convert equatorial to local horizontal coordinate*

Description

Convert from equatorial coordinates to local horizontal coordinates, i.e. azimuth and altitude. The method is taken from equations 8.5 and 8.6 of [1], or, equivalently, from equations 12.5 and 12.6 of [2].

Usage

```
equatorialToLocalHorizontal(rightAscension, declination, t, longitude, latitude)
```

Arguments

`rightAscension`	right ascension, e.g. calculated with `eclipticalToEquatorial`.
`declination`	declination, e.g. calculated with `eclipticalToEquatorial`.
`t`	time of observation.
`longitude`	longitude of observation, positive in eastern hemisphere.
`latitude`	latitude of observation, positive in northern hemisphere.

Value

A data frame containing columns `altitude` (angle above horizon, in degrees) and `azimuth` (angle anticlockwise from south, in degrees).

Author(s)

Dan Kelley, based on formulae in [1] and [2].

References

1. Meeus, Jean, 1982. Astronomical formulae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages.

2. Meeus, Jean, 1991. Astronomical algorithms. Willmann-Bell, Richmond VA, USA. 429 pages.

See Also

Other things related to astronomy: `eclipticalToEquatorial`, `julianCenturyAnomaly`, `julianDay`, `moonAngle`, `siderealTime`, `sunAngle` errorbars *Draw error bars on an existing xy diagram*

Description

Draw error bars on an existing xy diagram

Usage

```
errorbars(x, y, xe, ye, percent = FALSE, style = 0, length = 0.025, ...)
```

Arguments

`x`	x coordinates of points on the existing plot.
`y`	y coordinates of points on the existing plot.
`xe`	error on x coordinates of points on the existing plot, either a single number or a vector of length identical to that of `y`.
`ye`	as `xe` but for y coordinate.
`percent`	boolean flag indicating whether `xe` and `ye` are in terms of percent of the corresponding `x` and `y` values.
`style`	indication of the style of error bar. Using `style=0` yields simple line segments (drawn with `segments`) and `style=1` yields line segments with short perpendicular endcaps.
`length`	length of endcaps, for `style=1` only; it is passed to `arrows`, which is used to draw that style of error bars.
`...`	graphical parameters passed to the code that produces the error bars, e.g. to `segments` for `style=0`.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
S <- ctd[["salinity"]]
T <- ctd[["temperature"]]
plot(S, T)
errorbars(S, T, 0.05, 0.5)
```

fillGap *Fill a gap in an oce object*

Description

Sequences of NA values, are filled by linear interpolation between the non-NA values that bound the gap.

Usage

```
fillGap(x, method = c("linear"), rule = 1)
```

Arguments

x	an oce object.
method	to use; see “Details”.
rule	integer controlling behaviour at start and end of x. If rule=1, NA values at the ends are left in the return value. If rule=2, they are replaced with the nearest non-NA point.

Value

A new oce object, with gaps removed.

Bugs

1. Eventually, this will be expanded to work with any oce object. But, for now, it only works for vectors that can be coerced to numeric.
2. If the first or last point is NA, then x is returned unaltered.
3. Only method linear is permitted now.

Author(s)

Dan Kelley

Examples

```
library(oce)
# Integers
x <- c(1:2, NA, NA, 5:6)
y <- fillGap(x)
print(data.frame(x,y))
# Floats
x <- x + 0.1
y <- fillGap(x)
print(data.frame(x,y))
```

findBottom — *Find the Ocean Bottom in an Echosounder Object*

Description

Finds the depth in a Biosonics echosounder file, by finding the strongest reflector and smoothing its trace.

Usage

```
findBottom(x, ignore = 5, clean = despike)
```

Arguments

x	an object of class echosounder
ignore	number of metres of data to ignore, near the surface
clean	a function to clean the inferred depth of spikes

Value

A list with elements: the `time` of a ping, the `depth` of the inferred depth in metres, and the `index` of the inferred bottom location, referenced to the object's `depth` vector.

Author(s)

Dan Kelley

See Also

The documentation for `echosounder-class` explains the structure of `echosounder` objects, and also outlines the other functions dealing with them.

Other things related to echosounder data: `[[,echosounder-method`, `[[<-,echosounder-method`, `as.echosounder`, `echosounder-class`, `echosounder`, `plot,echosounder-method`, `read.echosounder`, `subset,echosounder-method`, `summary,echosounder-method`

findInOrdered — *Find indices of times in an ordered vector [deprecated]*

Description

WARNING: This function will be removed soon; see oce-deprecated. The replacement is trivial: just change a call like e.g. `findInOrdered(x,f)` to `findInterval(f,x)` (which is what the function started doing, on 2017-09-07, after a major bug was found).

Usage

```
findInOrdered(x, f)
```

Arguments

x	a numeric vector, in increasing order by value.
f	a numeric vector of items whose indices are sought.

Details

The indices point to the largest items in `x` that are less than or equal the values in `f`. This works by simply calling `findInterval(x=f, vec=x)`, and users are probably better off using `findInterval` directly.

Value

A numerical vector indicating the indices of left-sided neighbors.

Author(s)

Dan Kelley

Examples

```
findInOrdered(seq(0, 10, 1), c(1.2, 7.3))
```

---

firstFinite *Get first finite value in a vector or array, or NULL if none*

Description

Get first finite value in a vector or array, or NULL if none

Usage

```
firstFinite(v)
```

Arguments

v	A numerical vector or array.

`formatCI` *Confidence interval in parenthetic notation*

Description

Format a confidence interval in parenthetic notation.

Usage

```
formatCI(ci, style = c("+/-", "parentheses"), model, digits = NULL)
```

Arguments

`ci`	optional vector of length 2 or 3.
`style`	string indicating notation to be used.
`model`	optional regression model, e.g. returned by `lm` or `nls`.
`digits`	optional number of digits to use; if not supplied, `getOption("digits")` is used.

Details

If a `model` is given, then `ci` is ignored, and a confidence interval is calculated using `confint` with `level` set to 0.6914619. This `level` corresponds to a range of plus or minus one standard deviation, for the t distribution and a large number of degrees of freedom (since `qt(0.6914619, 100000)` is 0.5).

If `model` is missing, `ci` must be provided. If it contains 3 elements, then first and third elements are taken as the range of the confidence interval (which by convention should use the `level` stated in the previous paragraph), and the second element is taken as the central value. Alternatively, if `ci` has 2 elements, they are taken to be bounds of the confidence interval and their mean is taken to be the central value.

In the `+/-` notation, e.g. $a \pm b$ means that the true value lies between $a - b$ and $a + b$ with a high degree of certainty. Mills et al. (1993, section 4.1 on page 83) suggest that b should be set equal to 2 times the standard uncertainty or standard deviation. JCGM (2008, section 7.2.2 on pages 25 and 26), however, suggest that b should be set to the standard uncertainty, while also recommending that the $\pm$ notation be avoided altogether.

The `parentheses` notation is often called the compact notation. In it, the digits in parentheses indicate the uncertainty in the corresponding digits to their left, e.g. 12.34(3) means that the last digit (4) has an uncertainty of 3. However, as with the $\pm$ notation, different authorities offer different advice on defining this uncertainty; Mills et al. (1993, section 4.1 on page 83) provide an example in which the parenthetic notation has the same value as the $\pm$ notation, while JCM (2008, section 7.2.2 on pages 25 and 26) suggest halving the number put in parentheses.

The `foramtci` function is based on the JCM (2008) notation, i.e. `formatCI(ci=c(8,12), style="+/-")` yields `"10+-2"`, and `formatCI(ci=c(8,12), style="parentheses")` yields `"10(2)"`.

Note: if the confidence range exceeds the value, the `parentheses` format reverts to `+/-` format.

Value

If `ci` is given, the result is a character string with the estimate and its uncertainty, in plus/minus or parenthetic notation. If `model` is given, the result is a 1-column matrix holding character strings, with row names corresponding to the parameters of the model.

Author(s)

Dan Kelley

References

JCGM, 2008. *Evaluation of measurement data - Guide to the expression of uncertainty in measurement (JCGM 100:2008)*, published by the Joint Committee for Guides in Metrology. [http://www.bipm.org/en/publications/guides/gum.html] (See section 7.2.2 for a summary of notation, which shows equal values to the right of a `+-` sign and in parentheses.)

I. Mills, T. Cvitas, K. Homann, N. Kallay, and K. Kuchitsu, 1993. *Quantities, Units and Symbols in Physical Chemistry*, published Blackwell Science for the International Union of Pure and Applied Chemistry. (See section 4.1, page 83, for a summary of notation, which shows that a value to the right of a `+-` sign is to be halved if put in

Examples

```
x <- seq(0, 1, length.out=300)
y <- rnorm(n=300, mean=10, sd=1) * x
m <- lm(y~x)
print(formatCI(model=m))
```

`formatPosition` *Format Geographical Position in Degrees and Minutes*

Description

Format geographical positions to degrees, minutes, and hemispheres

Usage

```
formatPosition(latlon, isLat = TRUE, type = c("list", "string",
  "expression"), showHemi = TRUE)
```

Arguments

`latlon`	a vector of latitudes or longitudes
`isLat`	a boolean that indicates whether the quantity is latitude or longitude
`type`	a string indicating the type of return value (see below)
`showHemi`	a boolean that indicates whether to indicate the hemisphere

Value

A list containing `degrees`, `minutes`, `seconds`, and `hemispheres`, or a vector of strings or (broken) a vector of expressions.

Author(s)

Dan Kelley

Examples

```
library(oce)
formatPosition(10+1:10/60+2.8/3600)
formatPosition(10+1:10/60+2.8/3600, type="string")
```

`fullFilename` *Full name of file, including path*

Description

Determines the full name of a file, including the path. Used by many `read.X` routines, where `X` is the name of a class of object. This is a wrapper around `normalizePath`, with warnings turned off so that messages are not printed for unfound files (e.g. URLs).

Usage

```
fullFilename(filename)
```

Arguments

`filename`	name of file

Value

Full file name

Author(s)

Dan Kelley

`g1sst-class` *Class to Hold G1SST Satellite-model Data*

Description

G1SST is an acronym for global 1-km sea surface temperature, a product that combines satellite data with the model output. It is provided by the JPO ROMS (Regional Ocean Modelling System) modelling group. See the JPL website [1] to learn more about the data, and see the `read.g1sst` documentation for an example of downloading and plotting.

Details

It is important not to regard G1SST data in the same category as, say, `amsr-class` data, because the two products differ greatly with respect to cloud cover. The satellite used by `amsr-class` has the ability to sense water temperature even if there is cloud cover, whereas `g1sst` fills in cloud gaps with model simulations. It can be helpful to consult [1] for a given time, clicking and then unclicking the radio button that turns off the model-based filling of cloud gaps.

Author(s)

Dan Kelley

References

1. JPO OurOcean Portal `http://ourocean.jpl.nasa.gov/SST/` (link worked in 2016 but was seen to fail 2017 Feb 2).

See Also

Other things related to satellite data: `plot,satellite-method`, `read.g1sst`, `satellite-class`, `summary,satellite-method`

`geodDist` *Compute Geodesic Distance on Surface of Earth*

Description

This calculates geodesic distance between points on the earth, i.e. distance measured along the (presumed ellipsoidal) surface. The method involves the solution of the geodetic inverse problem, using T. Vincenty's modification of Rainsford's method with Helmert's elliptical terms.

Usage

```
geodDist(longitude1, latitude1 = NULL, longitude2 = NULL,
  latitude2 = NULL, alongPath = FALSE)
```

Arguments

`longitude1`	longitude or a vector of longitudes, **or** a `section` object, from which longitude and latitude are extracted and used instead of the next three arguments
`latitude1`	latitude or vector of latitudes (ignored if `longitude1` is a `section` object)
`longitude2`	optional longitude or vector of longitudes (ignored if `alongPath=TRUE`)
`latitude2`	optional latitude or vector of latitudes (ignored if `alongPath=TRUE`)
`alongPath`	boolean indicating whether to compute distance along the path, as opposed to distance from the reference point. If `alongPath=TRUE`, any values provided for `latitude2` and `longitude2` will be ignored.

Details

The function may be used in several different ways.

Case 1: `longitude1` is a `section` object. The values of `latitude1`, `longitude2`, and `latitude2` arguments are ignored, and the behaviour depends on the value of the `alongPath` argument. If `alongPath=FALSE`, the return value contains the geodetic distances of each station from the first one. If `alongPath=TRUE`, the return value is the geodetic distance along the path connecting the stations, in the order in which they are stored in the section.

Case 2: `longitude1` is a vector. If `longitude2` and `latitude2` are not given, then the return value is a vector containing the distances of each point from the first one, *or* the distance along the path connecting the points, according to the value of `alongPath`. On the other hand, if both `longitude2` and `latitude2` are specified, then the return result depends on the length of these arguments. If they are each of length 1, then they are taken as a reference point, from which the distances to `longitude1` and `latitude1` are calculated (ignoring the value of `alongPath`). However, if they are of the same length as `longitude1` and `latitude1`, then the return value is the distance between corresponding (`longitude1`,`latitude1`) and (`longitude2`,`latitude2`) values.

Value

Vector of distances in kilometres.

Author(s)

Dan Kelley based this on R code sent to him by Darren Gillis, who in 2003 had modified Fortran code that, according to comments in the source, had been written in 1974 by L. Pfeifer and J. G. Gergen.

References

T. Vincenty, "Direct and Inverse Solutions of Ellipsoid on the Ellipsoid with Application of Nested Equations", *Survey Review*, April 1975. (As of early 2009, this document is available at `http://www.ngs.noaa.gov/PUBS_LIB/inverse.pdf`.)

See Also

`geodXy`

Other functions relating to geodesy: `geodGc`, `geodXyInverse`, `geodXy`

Examples

```
library(oce)
km <- geodDist(100, 45, 100, 46)
data(section)
geodDist(section)
geodDist(section, alongPath=TRUE)
```

---

geodGc *Great-circle Segments Between Points on Earth*

Description

Each pair in the `longitude` and `latitude` vectors is considered in turn. For long vectors, this may be slow.

Usage

```
geodGc(longitude, latitude, dmax)
```

Arguments

`longitude`	vector of longitudes, in degrees east
`latitude`	vector of latitudes, in degrees north
`dmax`	maximum angular separation to tolerate between sub-segments, in degrees.

Value

Data frame of `longitude` and `latitude`.

Author(s)

Dan Kelley, based on code from Clark Richards, in turn based on formulae provided by Ed Williams [1].

References

1. `http://williams.best.vwh.net/avform.htm#Intermediate` (link worked for years but failed 2017-01-16).

See Also

Other functions relating to geodesy: `geodDist`, `geodXyInverse`, `geodXy`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, type='l',
        longitudelim=c(-80,10), latitudelim=c(35,80),
        projection="+proj=ortho", orientation=c(35, -35, 0))
## Great circle from New York to Paris (Lindberg's flight)
l <- geodGc(c(-73.94,2.35), c(40.67,48.86), 1)
mapLines(l$longitude, l$latitude, col='red', lwd=2)

## End(Not run)
```

---

geodXy — *Convert From Geographical to Geodesic Coordinates*

Description

The method, which may be useful in determining coordinate systems for a mooring array or a ship transects, calculates (x,y) from distance calculations along geodesic curves. See “Caution”.

Usage

```
geodXy(longitude, latitude, longitudeRef, latitudeRef,
  debug = getOption("oceDebug"))
```

Arguments

`longitude, latitude`
: vector of longitude and latitude

`longitudeRef, latitudeRef`
: numeric reference location. Poor results will be returned if these values are not close to the locations described by `longitude` and `latitude`. A sensible approach might be to set `longitudeRef` to `longitude[1]`, etc.

`debug`
: an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

The calculation is as follows. Consider the `i`-th point in the `longitude` and `latitude` vectors. To calculate `x[i]`, `geodDist` is used is to find the distance *along a geodesic curve* connecting (`longitude[i]`, `latitude[i]`) with (`longitudeRef`, `latitude[i]`). The resultant distance is multiplied by -1 if `longitude[i]-longitudeRef` is negative, and the result is assigned to `x[i]`. A similar procedure is used for `y[i]`.

Value

geodXy returns a data frame of x and y, geodesic distance components, measured in metres.

Caution

This scheme is without known precedent in the literature, and users should read the documentation carefully before deciding to use it.

Change history

On 2015-11-02, the names of the arguments were changed from lon, etc., to longitude, etc., to be in keeping with other oce functions.

On 2017-04-05, four changes were made. 1. Default values of longitudeRef and latitudeRef were removed, since the old defaults were inappropriate to most work. 2. The argument called rotate was eliminated, because it only made sense if the mean resultant x and y were zero. 3. The example was made more useful. 4. Pointers were made to lonlat2utm, which may be more useful.

Author(s)

Dan Kelley

See Also

geodDist

Other functions relating to geodesy: geodDist, geodGc, geodXyInverse

Examples

```
## Not run:
# Develop a transect-based axis system for final data(section) stations
library(oce)
data(section)
lon <- tail(section[["longitude", "byStation"]], 26)
lat <- tail(section[["latitude", "byStation"]], 26)
lonR <- tail(lon, 1)
latR <- tail(lat, 1)
data(coastlineWorld)
mapPlot(coastlineWorld, proj="+proj=merc",
        longitudelim=c(-75,-65), latitudelim=c(35,43), col="gray")
mapPoints(lon, lat)
XY <- geodXy(lon,lat,mean(lon), mean(lat))
angle <- 180/pi*atan(coef(lm(y~x, data=XY))[2])
mapCoordinateSystem(lonR, latR, 500, angle, col=2)
# Compare UTM calculation
UTM <- lonlat2utm(lon, lat, zone=18) # we need to set the zone for this task!
angleUTM <- 180/pi*atan(coef(lm(northing~easting, data=UTM))[2])
mapCoordinateSystem(lonR, latR, 500, angleUTM, col=3)
legend("topright", lwd=1, col=2:3, bg="white", title="Axis Rotation Angle",
       legend=c(sprintf("geod: %.1f deg", angle), sprintf("utm: %.1f deg",angleUTM)))

## End(Not run)
```

geodXyInverse *Inverse Geodesic Calculation*

Description

The calculation is done by finding a minimum value of a cost function that is the vector difference between (x,y) and the corresponding values returned by geodXy. See “Caution”.

Usage

```
geodXyInverse(x, y, longitudeRef, latitudeRef, debug = getOption("oceDebug"))
```

Arguments

x	value of x in metres, as given by geodXy
y	value of y in metres, as given by geodXy
longitudeRef	reference longitude, as supplied to geodXy
latitudeRef	reference latitude, as supplied to geodXy
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

The minimum is calculated in C for speed, using the nmmin function that is the underpinning for the Nelder-Meade version of the R function optim. If you find odd results, try setting debug=1 and rerunning, to see whether this optimizer is having difficulty finding a minimum of the mismatch function.

Value

a data frame containing longitude and latitude

Caution

This scheme is without known precedent in the literature, and users should read the documentation carefully before deciding to use it.

See Also

Other functions relating to geodesy: geodDist, geodGc, geodXy

GMTOffsetFromTz *Determine time offset from timezone*

Description

The data are from http://www.timeanddate.com/library/abbreviations/timezones/ and were hand-edited to develop this code, so there may be errors. Also, note that some of these contradict; if you examine the code, you'll see some commented-out portions that represent solving conflicting definitions by choosing the more common timezone abbreviation over a the less common one.

Usage

```
GMTOffsetFromTz(tz)
```

Arguments

tz	a timezone, e.g. UTC.

Value

Number of hours in offset, e.g. AST yields 4.

Author(s)

Dan Kelley

See Also

Other functions relating to time: decodeTime

Examples

```
library(oce)
cat("Atlantic Standard Time is ", GMTOffsetFromTz("AST"), "hours after UTC")
```

gps-class *Class to Store GPS Data*

Description

Class to store gps data. These objects may be read with read.gps or assembled with as.gps.

Author(s)

Dan Kelley

See Also

Other things related to gps data: `[[,gps-method`, `[[<-,gps-method`, `as.gps`, `plot,gps-method`, `read.gps`, `summary,gps-method`

---

grad — *Calculate the grad of a matrix by first differences*

Description

In the interior of the matrix, centred second-order differences are used to infer the components of the grad. Along the edges, first-order differences are used.

Usage

```
grad(h, x, y)
```

Arguments

h	a matrix
x	x values
y	y values

Value

A list containing gx and gy, matrices of the same dimension as `h`.

Author(s)

Dan Kelley, based on advice of Clark Richards, and mimicking a matlab function.

See Also

Other functions relating to vector calculus: `curl`

Examples

```
## Geostrophic flow around an eddy
library(oce)
dx <- 5e3
dy <- 10e3
x <- seq(-200e3, 200e3, dx)
y <- seq(-200e3, 200e3, dy)
R <- 100e3
h <- outer(x, y, function(x, y) 500*exp(-(x^2+y^2)/R^2))
grad <- grad(h, x, y)
par(mfrow=c(2, 2), mar=c(3, 3, 1, 1), mgp=c(2, 0.7, 0))
contour(x,y,h,asp=1, main=expression(h))
f <- 1e-4
```

```
gprime <- 9.8 * 1 / 1024
u <- -(gprime / f) * grad$gy
v <-  (gprime / f) * grad$gx
contour(x, y, u, asp=1, main=expression(u))
contour(x, y, v, asp=1, main=expression(v))
contour(x, y, sqrt(u^2+v^2), asp=1, main=expression(speed))
```

`gravity` *Acceleration due to earth gravity*

Description

Compute g, the acceleration due to gravity, as a function of latitude.

Usage

```
gravity(latitude = 45, degrees = TRUE)
```

Arguments

`latitude`	Latitude in $^\circ$N or radians north of the equator.
`degrees`	Flag indicating whether degrees are used for latitude; if set to `FALSE`, radians are used.

Details

Value not verified yet, except roughly.

Value

Acceleration due to gravity $[m^2/s]$.

Author(s)

Dan Kelley

References

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

Caution: Fofonoff and Millard (1983 UNESCO) use a different formula.

Examples

```
g <- gravity(45) # 9.8
```

handleFlags *Handle flags in oce objects*

Description

Data-quality flags are stored in the `metadata` slot of `oce-class` objects in a `list` named `flags`. The present function (a generic that has specialized versions for various data classes) provides a way to manipulate the core data based on the data-quality flags. For example, a common operation is to replace suspicious or erroneous data with `NA`.

If `metadata$flags` in the object supplied as the first argument is empty, then that object is returned, unaltered. Otherwise, `handleFlags` analyses the data-quality flags within the object, in relation to the `flags` argument, and interprets the `action` argument to select an action to be applied to matched data.

Reasonable defaults are used if `flags` and `actions` are not supplied (see 'Details'), but different schemes are used in different data archives, so it is risky to rely on these defaults. It is usually necessary to tailor `flags` and `actions` to the data and the analysis goals.

Usage

```
handleFlags(object, flags = NULL, actions = NULL,
  debug = options("oceDebug"))
```

Arguments

`object` An object of `oce`.

`flags` An optional `list` containing (a) items with names of entries in the `data` slot of `object`, or (b) a single unnamed item. In the first case, the attention is focussed on the named items, while in the second case the all the data in the `object`'s `data` slot are examined. Each element in the list must be set to an integer or vector of integers, specifying conditions to be met before actions are to be taken. See "Details" for the default that is used if `flags` is not supplied.

`actions` An optional `list` that contains items with names that match those in the `flags` argument. If `actions` is not supplied, the default will be to set all values identified by `flags` to `NA`; this can also be specified by specifying `actions=list("NA")`. It is also possible to specify functions that calculate replacement values. These are provided with `object` as the single argument, and must return a replacement for the data item in question. See "Details" for the default that is used if `actions` is not supplied.

`debug` An optional integer specifying the degree of debugging, with value 0 meaning to skip debugging and 1 or higher meaning to print some information about the arguments and the data. It is usually a good idea to set this to 1 for initial work with a dataset, to see which flags are being handled for each data item. If not supplied, this defaults to the value of `getOption("oceDebug")`.

Details

Each specialized variant of this function has its own defaults for `flags` and `actions`.

See Also

Other functions that handle data-quality flags: `handleFlags,argo-method`, `handleFlags,ctd-method`, `handleFlags,section-method`

`handleFlags,argo-method` *Handle Flags in ARGO Objects*

Description

Data-quality flags are stored in the `metadata` slot of `oce-class` objects in a `list` named `flags`. The present function (a generic that has specialized versions for various data classes) provides a way to manipulate the core data based on the data-quality flags. For example, a common operation is to replace suspicious or erroneous data with `NA`.

If `metadata$flags` in the object supplied as the first argument is empty, then that object is returned, unaltered. Otherwise, `handleFlags` analyses the data-quality flags within the object, in relation to the `flags` argument, and interprets the `action` argument to select an action to be applied to matched data.

Reasonable defaults are used if `flags` and `actions` are not supplied (see 'Details'), but different schemes are used in different data archives, so it is risky to rely on these defaults. It is usually necessary to tailor `flags` and `actions` to the data and the analysis goals.

Usage

```
## S4 method for signature 'argo'
handleFlags(object, flags = list(), actions = list(),
  debug = integer())
```

Arguments

`object`	An object of `argo-class`.
`flags`	An optional `list` containing (a) items with names of entries in the `data` slot of `object`, or (b) a single unnamed item. In the first case, the attention is focussed on the named items, while in the second case the all the data in the `object`'s `data` slot are examined. Each element in the list must be set to an integer or vector of integers, specifying conditions to be met before actions are to be taken. See "Details" for the default that is used if `flags` is not supplied.
`actions`	An optional `list` that contains items with names that match those in the `flags` argument. If `actions` is not supplied, the default will be to set all values identified by `flags` to `NA`; this can also be specified by specifying `actions=list("NA")`. It is also possible to specify functions that calculate replacement values. These

are provided with `object` as the single argument, and must return a replacement for the data item in question. See "Details" for the default that is used if `actions` is not supplied.

debug
: An optional integer specifying the degree of debugging, with value 0 meaning to skip debugging and 1 or higher meaning to print some information about the arguments and the data. It is usually a good idea to set this to 1 for initial work with a dataset, to see which flags are being handled for each data item. If not supplied, this defaults to the value of `getOption("oceDebug")`.

Details

If `flags` and `actions` are not provided, the default is to use ARGO flags [1], in which the value 1 indicates good data, and other values indicate either unchecked, suspicious, or bad data. Any data not flagged as good are set to `NA` in the returned value. Since Argo flag codes run from 0 to 9, with 1 indicating the highest level of confidence in the data, the defaults are `flags=list(c(0,2:9))` and `actions=list("NA")`.

References

1. http://www.argo.ucsd.edu/Argo_date_guide.html#dmodedata

See Also

Other functions that handle data-quality flags: `handleFlags,ctd-method`, `handleFlags,section-method`, `handleFlags`

Other things related to argo data: `[[,argo-method`, `[[<-,argo-method`, `argo-class`, `argoGrid`, `argoNames2oceNames`, `argo`, `as.argo`, `plot,argo-method`, `read.argo`, `subset,argo-method`, `summary,argo-method`

Examples

```
## Not run:
library(oce)
data(argo)
# 1. Default: anything not flagged as 1 is set to NA, to focus
# solely on 'good', in the Argo scheme.
argoNew <- handleFlags(argo)
# demonstrate replacement, looking at the second profile
f <- argo[["salinityFlag"]][,2] # first column with a flag=4 entry
df <- data.frame(flag=f, orig=argo[["salinity"]][,2], new=argoNew[["salinity"]][,2])
df[11:15,]
##    flag   orig    new
## 11    1 35.207 35.207
## 12    1 35.207 35.207
## 13    4 35.209     NA
## 14    1 35.207 35.207
## 15    1 35.207 35.207

# 2. A less restrictive case: include also 'questionable' data,
# and only apply this action to salinity.
```

```
argoNew <- handleFlags(argo, flags=list(salinity=4))

## End(Not run)
```

`handleFlags,ctd-method` *Handle Flags in CTD Objects*

Description

Data-quality flags are stored in the `metadata` slot of `oce-class` objects in a `list` named `flags`. The present function (a generic that has specialized versions for various data classes) provides a way to manipulate the core data based on the data-quality flags. For example, a common operation is to replace suspicious or erroneous data with `NA`.

If `metadata$flags` in the object supplied as the first argument is empty, then that object is returned, unaltered. Otherwise, `handleFlags` analyses the data-quality flags within the object, in relation to the `flags` argument, and interprets the `action` argument to select an action to be applied to matched data.

Reasonable defaults are used if `flags` and `actions` are not supplied (see 'Details'), but different schemes are used in different data archives, so it is risky to rely on these defaults. It is usually necessary to tailor `flags` and `actions` to the data and the analysis goals.

Usage

```
## S4 method for signature 'ctd'
handleFlags(object, flags = list(), actions = list(),
  debug = integer())
```

Arguments

`object` A `ctd` object, i.e. one inheriting from `ctd-class`.

`flags` An optional `list` containing (a) items with names of entries in the `data` slot of `object`, or (b) a single unnamed item. In the first case, the attention is focussed on the named items, while in the second case the all the data in the `object`'s `data` slot are examined. Each element in the list must be set to an integer or vector of integers, specifying conditions to be met before actions are to be taken. See "Details" for the default that is used if `flags` is not supplied.

`actions` An optional `list` that contains items with names that match those in the `flags` argument. If `actions` is not supplied, the default will be to set all values identified by `flags` to `NA`; this can also be specified by specifying `actions=list("NA")`. It is also possible to specify functions that calculate replacement values. These are provided with `object` as the single argument, and must return a replacement for the data item in question. See "Details" for the default that is used if `actions` is not supplied.

debug — An optional integer specifying the degree of debugging, with value 0 meaning to skip debugging and 1 or higher meaning to print some information about the arguments and the data. It is usually a good idea to set this to 1 for initial work with a dataset, to see which flags are being handled for each data item. If not supplied, this defaults to the value of getOption("oceDebug").

Details

If flags and actions are not provided, the default is to use WHP (World Hydrographic Program) flags [1], in which the value 2 indicates good data, and other values indicate either unchecked, suspicious, or bad data. Any data not flagged as good are set to NA in the returned value. (An exception is for salinity: if the item named salinity has a bad flag but salinityBottle has a good flag, then the bottle value is substituted, and a warning is issued.) Since WHP flag codes run from 1 to 9, this default is equivalent to setting flags=list(all=c(1, 3:9)) along with action=list("NA").

References

1. https://www.nodc.noaa.gov/woce/woce_v3/wocedata_1/whp/exchange/exchange_format_desc.htm

See Also

Other functions that handle data-quality flags: handleFlags,argo-method, handleFlags,section-method, handleFlags

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
library(oce)
data(section)
stn <- section[["station", 100]]
# 1. Default: anything not flagged as 2 is set to NA, to focus
# solely on 'good', in the World Hydrographic Program scheme.
STN <- handleFlags(stn)
data.frame(old=stn[['salinity']], flag=stn[['salinityFlag']], new=STN[['salinity']])

# 2. A less restrictive case: include also 'questionable' data,
# and only apply this action to salinity.
STN <- handleFlags(stn, flags=list(salinity=c(1, 4:9)))

# 3. A Canadian Department of Fisheries and Oceans convention for
# some of its data files lists flags as 0=unchecked, 1=good,
# 2=uncertain, 3=doubtful, 4=wrong, and 5=changed, so a
# trusting arrangement would be to discard 2:4, and a more
# cautious approach would be to also discard 0.
STN <- handleFlags(stn, flags=list(2:4))
```

```
STN <- handleFlags(stn, flags=list(c(0,2:4)))

# 4. Use smoothed TS relationship to nudge questionable data.
# This is not a recommended procedure, but rather just a simple
# illustration of how to supply a function for an action.
f<-function(x) {
  S <- x[["salinity"]]
  T <- x[["temperature"]]
  df <- 0.5 * length(S) # smooths a bit
  sp <- smooth.spline(T, S, df=df)
  0.5 * (S + predict(sp, T)$y)
}
par(mfrow=c(1,2))
STN <- handleFlags(stn, flags=list(salinity=c(1,3:9)), action=list(salinity=f))
plotProfile(stn, "salinity", mar=c(3, 3, 3, 1))
p <- stn[['pressure']]
par(mar=c(3, 3, 3, 1))
plot(STN[['salinity']] - stn[['salinity']], p, ylim=rev(range(p)))
```

`handleFlags,section-method`

Handle flags in Section Objects

Description

Data-quality flags are stored in the `metadata` slot of `oce-class` objects in a `list` named `flags`. The present function (a generic that has specialized versions for various data classes) provides a way to manipulate the core data based on the data-quality flags. For example, a common operation is to replace suspicious or erroneous data with `NA`.

If `metadata$flags` in the object supplied as the first argument is empty, then that object is returned, unaltered. Otherwise, `handleFlags` analyses the data-quality flags within the object, in relation to the `flags` argument, and interprets the `action` argument to select an action to be applied to matched data.

Reasonable defaults are used if `flags` and `actions` are not supplied (see 'Details'), but different schemes are used in different data archives, so it is risky to rely on these defaults. It is usually necessary to tailor `flags` and `actions` to the data and the analysis goals.

Usage

```
## S4 method for signature 'section'
handleFlags(object, flags = list(), actions = list(),
  debug = integer())
```

Arguments

`object`	An object of `section-class`.

flags	An optional list containing (a) items with names of entries in the data slot of object, or (b) a single unnamed item. In the first case, the attention is focussed on the named items, while in the second case the all the data in the object's data slot are examined. Each element in the list must be set to an integer or vector of integers, specifying conditions to be met before actions are to be taken. See "Details" for the default that is used if flags is not supplied.
actions	An optional list that contains items with names that match those in the flags argument. If actions is not supplied, the default will be to set all values identified by flags to NA; this can also be specified by specifying actions=list("NA"). It is also possible to specify functions that calculate replacement values. These are provided with object as the single argument, and must return a replacement for the data item in question. See "Details" for the default that is used if actions is not supplied.
debug	An optional integer specifying the degree of debugging, with value 0 meaning to skip debugging and 1 or higher meaning to print some information about the arguments and the data. It is usually a good idea to set this to 1 for initial work with a dataset, to see which flags are being handled for each data item. If not supplied, this defaults to the value of getOption("oceDebug").

Details

If flags and actions are not provided, the default is to use WHP (World Hydrographic Program) flags [1], in which the value 2 indicates good data, and other values indicate either unchecked, suspicious, or bad data. Any data not flagged as good are set to NA in the returned value. Since WHP flag codes run from 1 to 9, this default is equivalent to setting flags=list(all=c(1, 3:9)) along with action=list("NA").

References

1. https://www.nodc.noaa.gov/woce/woce_v3/wocedata_1/whp/exchange/exchange_format_desc.htm

See Also

Other functions that handle data-quality flags: handleFlags,argo-method, handleFlags,ctd-method, handleFlags

Other things related to section data: [[,section-method, [[<-,section-method, as.section, plot,section-method, read.section, section-class, sectionAddStation, sectionGrid, sectionSmooth, sectionSort, section, subset,section-method, summary,section-method

Examples

```
library(oce)
data(section)
section2 <- handleFlags(section)
par(mfrow=c(2, 1))
plotTS(section)
plotTS(section2)
```

handleFlags,vector-method *Signal erroneous application to non-oce objects*

Description

Signal erroneous application to non-oce objects

Usage

```
## S4 method for signature 'vector'
handleFlags(object, flags = list(), actions = list(),
  debug = integer())
```

Arguments

object	A vector, which cannot be the case for oce objects.
flags	Ignored.
actions	Ignored.
debug	Ignored.

head.oce *Extract The Start of an Oce Object*

Description

Extract The Start of an Oce Object

Usage

```
## S3 method for class 'oce'
head(x, n = 6L, ...)
```

Arguments

x	An oce object of a suitable class (presently only adp is permitted).
n	Number of elements to extract.
...	ignored

See Also

tail.oce, which yields the end of an oce object.

imagep *Plot an Image with a Color Palette*

Description

Plot an image with a colour palette, in a way that does not conflict with `par(mfrow)` or `layout`. To plot just a palette, e.g. to get an x-y plot with points coloured according to a palette, use `drawPalette` and then draw the main diagram.

Usage

```
imagep(x, y, z, xlim, ylim, zlim, zclip = FALSE, flipy = FALSE, xlab = "",
  ylab = "", zlab = "", zlabPosition = c("top", "side"),
  decimate = TRUE, breaks, col, colormap, labels = NULL, at = NULL,
  drawContours = FALSE, drawPalette = TRUE, drawTriangles = FALSE,
  tformat, drawTimeRange = getOption("oceDrawTimeRange"),
  filledContour = FALSE, missingColor = NULL, useRaster,
  mgp = getOption("oceMgp"), mar, mai.palette, xaxs = "i", yaxs = "i",
  asp = NA, cex = par("cex"), axes = TRUE, main = "", axisPalette,
  add = FALSE, debug = getOption("oceDebug"), ...)
```

Arguments

`x, y`	These have different meanings in different modes of operation. *Mode 1.* One mode has them meaning the locations of coordinates along which values matrix `z` are defined. In this case, both `x` and `y` must be supplied and, within each, the values must be finite and distinct; if values are out of order, they (and `z`) will be transformed to put them in order. ordered in a matching way). *Mode 2.* If `z` is provided but not `x` and `y`, then the latter are constructed to indicate the indices of the matrix, in contrast to the range of 0 to 1, as is the case for `image`. *Mode 3.* If `x` is a list, its components `x$x` and `x$y` are used for `x` and `y`, respectively. If the list has component `z` this is used for `z`. (NOTE: these arguments are meant to mimic those of `image`, which explains the same description here.) *Mode 4.* There are also some special cases, e.g. if `x` is a topographic object such as can be created with `read.topo` or `as.topo`, then longitude and latitude are used for axes, and topographic height is drawn.
`z`	A matrix containing the values to be plotted (NAs are allowed). Note that `x` can be used instead of `z` for convenience. (NOTE: these arguments are meant to mimic those of `image`, which explains the same description here.)
`xlim, ylim`	Limits on x and y axes.
`zlim`	If missing, the z scale is determined by the range of the data. If provided, `zlim` may take several forms. First, it may be a pair of numbers that specify the limits for the colour scale. Second, it could be the string `"histogram"`, to yield a flattened histogram (i.e. to increase contrast). Third, it could be the string `"symmetric"`, to yield limits that are symmetric about zero, which can be helpful in drawing velocity fields, for which a zero value has a particular meaning (in which case, a good colour scheme might be `col=oceColorsTwo`).

zclip
: Logical, indicating whether to clip the colours to those corresponding to `zlim`. This only works if `zlim` is provided. Clipped regions will be coloured with `missingColor`. Thus, clipping an image is somewhat analogous to clipping in an xy plot, with clipped data being ignored, which in an image means to be be coloured with `missingColor`.

flipy
: Logical, with `TRUE` indicating that the image should be flipped top to bottom (e.g. to produce a profile image for a downward-looking acoustic-doppler profile).

xlab, ylab, zlab
: Names for x axis, y axis, and the image values.

zlabPosition
: String indicating where to put the label for the z axis, either at the top-right of the main image, or on the side, in the axis for the palette.

decimate
: Controls whether the image will be decimated before plotting, in three possible cases. **Case 1.** If `decimate=FALSE` then every grid cell in the matrix will be represented by a pixel in the image. **Case 2 (the default).** If `decimate=TRUE`, then decimation will be done in the horizontal or vertical direction (or both) if the length of the corresponding edge of the `z` matrix exceeds 800. (This also creates a warning message.) The decimation factor is computed as the integer just below the ratio of `z` dimension to 400. Thus, no decimation is done if the dimension is less than 800, but if the dimension s between 800 and 1199, only every second grid point is mapped to a pixel in the image. **Case 3.** If `decimate` is an integer, then that `z` is subsampled at `seq.int(1L, dim(z)[1], by=decimate)` (as is `x`), and the same is done for the `y` direction. **Case 4.** If `decimate` is a vector of two integers, the first is used for the first index of `z`, and the second is used for the second index.

breaks
: The `z` values for breaks in the colour scheme. If this is of length 1, the value indicates the desired number of breaks, which is supplied to `pretty`, in determining clean break points.

col
: Either a vector of colours corresponding to the breaks, of length 1 plus the number of breaks, or a function specifying colours, e.g. `oce.colorsJet` for a rainbow.

colormap
: A colour map as created by `colormap`. If provided, then `colormap$breaks` and `colormap$col` take precedence over the present arguments `breaks` and `col`. (All of the other contents of `colormap` are ignored, though.)

labels
: Optional vector of labels for ticks on palette axis (must correspond with `at`).

at
: Optional vector of positions for the `labels`.

drawContours
: Logical value indicating whether to draw contours on the image, and palette, at the colour breaks. Images with a great deal of high-wavenumber variation look poor with contours.

drawPalette
: Indication of the type of palette to draw, if any. If `drawPalette=TRUE`, a palette is drawn at the right-hand side of the main image. If `drawPalette=FALSE`, no palette is drawn, and the right-hand side of the plot has a thin margin. If `drawPalette="space"`, then no palette is drawn, but space is put in the right-hand margin to occupy the region in which the palette would have been drawn. This last form is useful for producing stacked plots with uniform left and right margins, but with palettes on only some of the images.

drawTriangles	Logical value indicating whether to draw triangles on the top and bottom of the palette. This is passed to drawPalette.
tformat	Optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
drawTimeRange	Logical, only used if the x axis is a time. If TRUE, then an indication of the time range of the data (not the axis) is indicated at the top-left margin of the graph. This is useful because the labels on time axes only indicate hours if the range is less than a day, etc.
filledContour	Boolean value indicating whether to use filled contours to plot the image.
missingColor	A colour to be used to indicate missing data, or NULL for transparent (to see this, try setting par("bg")<-"red").
useRaster	A logical value passed to image, in cases where filledContour is FALSE. Setting useRaster=TRUE can alleviate some anti-aliasing effects on some plot devices; see the documentation for image.
mgp	A 3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	A 4-element Value to be used with par("mar"). If not given, a reasonable value is calculated based on whether xlab and ylab are empty strings.
mai.palette	Palette margin corrections (in inches), added to the mai value used for the palette. Use with care.
xaxs	Character indicating whether image should extend to edge of x axis (with value "i") or not; see par("xaxs").
yaxs	As xaxs but for y axis.
asp	Aspect ratio of the plot, as for plot.default. If x inherits from topo-class and asp=NA (the default) then asp is redefined to be the reciprocal of the mean latitude in x, as a way to reduce geographical distortion. Otherwise, if asp is not NA, then it is used directly.
cex	Size of labels on axes and palette; see par("cex").
axes	Logical, set TRUE to get axes on the main image.
main	Title for plot.
axisPalette	Optional replacement function for axis(), passed to drawPalette.
add	Logical value indicating whether to add to an existing plot. The default value, FALSE indicates that a new plot is to be created. However, if add is TRUE, the idea is to add an image (but not its palette or its axes) to an existing plot. Clearly, then, arguments such xlim are to be ignored. Indeed, if add=TRUE, the only arguments examined are x (which must be a vector; the mode of providing a matrix or oce object does not work), y, z, decimate, plus either colormap or both breaks and col.
debug	A flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	Optional arguments passed to plotting functions.

Details

By default, creates an image with a colour palette to the right. The effect is similar to `filled.contour` except that with `imagep` it is possible to set the `layout` outside the function, which enables the creation of plots with many image-palette panels. Note that the contour lines may not coincide with the colour transitions, in the case of coarse images.

Note that this does not use `layout` or any of the other screen splitting methods. It simply manipulates margins, and draws two plots together. This lets users employ their favourite layout schemes.

NOTE: `imagep` is an analogue of `image`, and from that it borrows a the convention that the number of rows in the matrix corresponds to to `x` axis, not the `y` axis. (Actually, `image` permits the length of `x` to match either `nrow(z)` or `1+nrow(z)`, but here only the first is permitted.)

Value

A list is silently returned, containing `xat` and `yat`, values that can be used by `oce.grid` to add a grid to the plot.

Author(s)

Dan Kelley and Clark Richards

See Also

This uses `drawPalette`, and is used by `plot,adp-method`, `plot,landsat-method`, and other image-generating functions.

Examples

```
library(oce)

# 1. simplest use
imagep(volcano)

# 2. something oceanographic (internal-wave speed)
h <- seq(0, 50, length.out=100)
drho <- seq(1, 3, length.out=200)
speed <- outer(h, drho, function(drho, h) sqrt(9.8 * drho * h / 1024))
imagep(h, drho, speed, xlab="Equivalent depth [m]",
ylab=expression(paste(Delta*rho, " [kg/m^3]")),
zlab="Internal-wave speed [m/s]")

# 3. fancy labelling on atan() function
x <- seq(0, 1, 0.01)
y <- seq(0, 1, 0.01)
angle <- outer(x,y,function(x,y) atan2(y,x))
imagep(x, y, angle, filledContour=TRUE, breaks=c(0, pi/4, pi/2),
       col=c("lightgray", "darkgray"),
       at=c(0, pi/4, pi/2),
       labels=c(0, expression(pi/4), expression(pi/2)))

# 4. a colormap case
```

```
data(topoWorld)
cm <- colormap(name="gmt_globe")
imagep(topoWorld, colormap=cm)
```

---

integerToAscii   *Decode integer to corresponding ASCII code*

Description

Decode integer to corresponding ASCII code

Usage

```
integerToAscii(i)
```

Arguments

i	an integer, or integer vector.

Value

A character, or character vector.

Author(s)

Dan Kelley

Examples

```
library(oce)
A <- integerToAscii(65)
cat("A=", A, "\n")
```

---

integrateTrapezoid   *Trapezoidal Integration*

Description

Estimate the integral of one-dimensional function using the trapezoidal rule.

Usage

```
integrateTrapezoid(x, y, type = c("A", "dA", "cA"), xmin, xmax)
```

Arguments

- `x, y` — vectors of x and y values. In the normal case, these vectors are both supplied, and of equal length. There are also two special cases. First, if `y` is missing, then `x` is taken to be `y`, and a new `x` is constructed as `seq_along(y)`. Second, if `length(x)` is 1 and `length(y)` exceeds 1, then `x` is replaced by `x*seq_along(y)`.
- `type` — Flag indicating the desired return value (see "Value").
- `xmin, xmax` — Optional numbers indicating the range of the integration. These values may be used to restrict the range of integration, or to extend it; in either case, `approx` with `rule=2` is used to create new x and y vectors.

Value

If `type="A"` (the default), a single value is returned, containing the estimate of the integral of `y=y(x)`. If `type="dA"`, a numeric vector of the same length as `x`, of which the first element is zero, the second element is the integral between `x[1]` and `x[2]`, etc. If `type="cA"`, the result is the cumulative sum (as in `cumsum`) of the values that would be returned for `type="dA"`. See "Examples".

Bugs

There is no handling of `NA` values.

Author(s)

Dan Kelley

Examples

```
x <- seq(0, 1, length.out=10) # try larger length.out to see if area approaches 2
y <- 2*x + 3*x^2
A <- integrateTrapezoid(x, y)
dA <- integrateTrapezoid(x, y, "dA")
cA <- integrateTrapezoid(x, y, "cA")
print(A)
print(sum(dA))
print(tail(cA, 1))
print(integrateTrapezoid(diff(x[1:2]), y))
print(integrateTrapezoid(y))
```

`interpBarnes` *Grid data using Barnes algorithm*

Description

The algorithm follows that described by Koch et al. (1983), with the addition of the ability to blank out the grid in spots where data are sparse, using the `trim` argument, and the ability to pre-grid, with the `pregrid` argument.

Usage

```
interpBarnes(x, y, z, w, xg, yg, xgl, ygl, xr, yr, gamma = 0.5,
  iterations = 2, trim = 0, pregrid = FALSE,
  debug = getOption("oceDebug"))
```

Arguments

`x, y`	a vector of x and ylocations.
`z`	a vector of z values, one at each (x,y) location.
`w`	a optional vector of weights at the (x,y) location. If not supplied, then a weight of 1 is used for each point, which means equal weighting. Higher weights give data points more influence.
`xg, yg`	optional vectors defining the x and y grids. If not supplied, these values are inferred from the data, using e.g. `pretty(x, n=50)`.
`xgl, ygl`	optional lengths of the x and y grids, to be constructed with `seq` spanning the data range. These values `xgl` are only examined if `xg` and `yg` are not supplied.
`xr, yr`	optional values defining the width of the radius ellipse in the x and y directions. If not supplied, these are calculated as the span of x and y over the square root of the number of data.
`gamma`	grid-focussing parameter. At each iteration, `xr` and `yr` are reduced by a factor of `sqrt(gamma)`.
`iterations`	number of iterations.
`trim`	a number between 0 and 1, indicating the quantile of data weight to be used as a criterion for blanking out the gridded value (using `NA`). If 0, the whole `zg` grid is returned. If >0, any spots on the grid where the data weight is less than the `trim`-th `quantile` are set to `NA`. See examples.
`pregrid`	an indication of whether to pre-grid the data. If `FALSE`, this is not done, i.e. conventional Barnes interpolation is performed. Otherwise, then the data are first averaged within grid cells using `binMean2D`. If `pregrid` is `TRUE` or `4`, then this averaging is done within a grid that is 4 times finer than the grid that will be used for the Barnes interpolation. Otherwise, `pregrid` may be a single integer indicating the grid refinement (4 being the result if `TRUE` had been supplied), or a vector of two integers, for the grid refinement in x and y. The purpose of using `pregrid` is to speed processing on large datasets, and to remove spatial bias (e.g. with a single station that is repeated frequently in an otherwise seldom-sampled region). A form of pregridding is done in the World Ocean Atlas, for example.
`debug`	a flag that turns on debugging. Set to 0 for no debugging information, to 1 for more, etc; the value is reduced by 1 for each descendent function call.

Value

A list containing: `xg`, a vector holding the x-grid); `yg`, a vector holding the y-grid; `zg`, a matrix holding the gridded values; `wg`, a matrix holding the weights used in the interpolation at its final iteration; and `zd`, a vector of the same length as `x`, which holds the interpolated values at the data points.

Author(s)

Dan Kelley

References

S. E. Koch and M. DesJardins and P. J. Kocin, 1983. “An interactive Barnes objective map analysis scheme for use with satellite and conventional data,” *J. Climate Appl. Met.*, vol 22, p. 1487-1503.

See Also

See `wind`.

Examples

```
library(oce)

# 1. contouring example, with wind-speed data from Koch et al. (1983)
data(wind)
u <- interpBarnes(wind$x, wind$y, wind$z)
contour(u$xg, u$yg, u$zg, labcex=1)
text(wind$x, wind$y, wind$z, cex=0.7, col="blue")
title("Numbers are the data")

# 2. As 1, but blank out spots where data are sparse
u <- interpBarnes(wind$x, wind$y, wind$z, trim=0.1)
contour(u$xg, u$yg, u$zg, level=seq(0, 30, 1))
points(wind$x, wind$y, cex=1.5, pch=20, col="blue")

# 3. As 1, but interpolate back to points, and display the percent mismatch
u <- interpBarnes(wind$x, wind$y, wind$z)
contour(u$xg, u$yg, u$zg, labcex=1)
mismatch <- 100 * (wind$z - u$zd) / wind$z
text(wind$x, wind$y, round(mismatch), col="blue")
title("Numbers are percent mismatch between grid and data")


# 4. As 3, but contour the mismatch
mismatchGrid <- interpBarnes(wind$x, wind$y, mismatch)
contour(mismatchGrid$xg, mismatchGrid$yg, mismatchGrid$zg, labcex=1)

# 5. One-dimensional example, smoothing a salinity profile
data(ctd)
p <- ctd[["pressure"]]
y <- rep(1, length(p)) # fake y data, with arbitrary value
S <- ctd[["salinity"]]
pg <- pretty(p, n=100)
g <- interpBarnes(p, y, S, xg=pg, xr=1)
plot(S, p, cex=0.5, col="blue", ylim=rev(range(p)))
lines(g$zg, g$xg, col="red")
```

julianCenturyAnomaly *Julian-Day number to Julian century*

Description

Convert a Julian-Day number to a time in julian centuries since noon on January 1, 1900. The method follows Meese (1982 equation 15.1). The example reproduces the example provided by Meeuse (1982 example 15.a), with fractional error 3e-8.

Usage

```
julianCenturyAnomaly(jd)
```

Arguments

`jd`	a julian day number, e.g. as given by `julianDay`.

Value

Julian century since noon on January 1, 1900.

Author(s)

Dan Kelley

References

Meeus, Jean, 1982. Astronomical formuae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages

See Also

Other things related to astronomy: `eclipticalToEquatorial`, `equatorialToLocalHorizontal`, `julianDay`, `moonAngle`, `siderealTime`, `sunAngle`

Other things related to time: `ctimeToSeconds`, `julianDay`, `numberAsHMS`, `numberAsPOSIXct`, `secondsToCtime`, `unabbreviateYear`

Examples

```
t <- ISOdatetime(1978, 11, 13, 4, 35, 0, tz="UTC")
jca <- julianCenturyAnomaly(julianDay(t))
cat(format(t), "is Julian Century anomaly", format(jca, digits=8), "\n")
```

`julianDay` *Convert a POSIXt time to a Julian day*

Description

Convert a POSIXt time to a Julian day, using the method provided in Chapter 3 of Meeus (1982). It should be noted that Meeus and other astronomical treatments use fractional days, whereas the present code follows the R convention of specifying days in whole numbers, with hours, minutes, and seconds also provided as necessary. Conversion is simple, as illustrated in the example for 1977 April 26.4, for which Meeus calculates julian day 2443259.9. Note that the R documentation for `julian` suggests another formula, but the point of the present function is to match the other Meeus formulae, so that suggestion is ignored here.

Usage

```
julianDay(t, year = NA, month = NA, day = NA, hour = NA, min = NA,
  sec = NA, tz = "UTC")
```

Arguments

`t`	a time, in POSIXt format, e.g. as created by `as.POSIXct`, `as.POSIXlt`, or `numberAsPOSIXct`. If this is provided, the other arguments are ignored.
`year`	year, to be provided along with `month`, etc., if `t` is not provided.
`month`	month, numbered with January being 1.
`day`	day in month, starting at 1.
`hour`	hour of day.
`min`	minute of hour
`sec`	second of hour
`tz`	timezone

Value

A Julian-Day number, in astronomical convention as explained in Meeus.

Author(s)

Dan Kelley

References

Meeus, Jean, 1982. Astronomical formuae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages

See Also

Other things related to astronomy: `eclipticalToEquatorial`, `equatorialToLocalHorizontal`, `julianCenturyAnomaly`, `moonAngle`, `siderealTime`, `sunAngle`

Other things related to time: `ctimeToSeconds`, `julianCenturyAnomaly`, `numberAsHMS`, `numberAsPOSIXct`, `secondsToCtime`, `unabbreviateYear`

Examples

```
t <- ISOdatetime(1977, 4, 26, hour=0, min=0, sec=0, tz="ET")+0.4*86400
jd <- julianDay(t)
cat(format(t), "is Julian Day", format(jd, digits=14), "\n")
```

`ladp-class` *Class to store lowered-adp data*

Description

Class to store data measured with a lowered ADP (also known as ADCP) device.

Accessing data

Consider an object named `ladp`.

Metadata (contained in the S4 slot named `metadata`) may be retrieved or set by name, `ladp[["longitude"]] <- ladp[["lo` corrects a one-degree error. Use `names(ladp@metadata)` to find the names of the metadata.

Column data may be accessed by name, e.g. `ladp[["u"]]`, `ladp[["v"]]`, `ladp[["pressure"]]`, etc. There may also be columns for `"temperature"` and `"salinity"`, and possibly other things. Use `names(ladp@data)` to find the names of the data.

Author(s)

Dan Kelley

See Also

Other things related to `ladp` data: `[[,ladp-method`, `as.ladp`, `plot,ladp-method`, `summary,ladp-method`

`landsat` *Sample landsat Dataset*

Description

This is a subset of the Landsat-8 image designated LC80080292014065LGN00, an image from March 2014 that covers Nova Scotia and portions of the Bay of Fundy and the Scotian Shelf. The image is decimated to reduce the memory requirements of this package, yielding a spatial resolution of about 2km.

Details

The original data were downloaded from the USGS earthexplorer website, although other sites can also be used to uncover it by name. The original data were decimation by a factor of 100 to reduce the file size from about 1GB to under 100Kb.

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to `landsat` data: `[[,landsat-method`, `landsat-class`, `landsatAdd`, `landsatTrim`, `plot,landsat-method`, `read.landsat`, `summary,landsat-method`

`landsat-class` *Class to Hold landsat Data*

Description

This class has the standard slots of an `oce-class` object. Landsat data are available at several websites (e.g. [1]). Although the various functions may work for other satellites, the discussion here focusses on Landsat 8 and Landsat 7.

Data storage

The data are stored with 16-bit resolution. Oce breaks these 16 bits up into most-significant and least-significant bytes. For example, the aerosol band of a Landsat object named `x` are contained within `x@data$aerosol$msb` and `x@data$aerosol$lsb`, each of which is a matrix of raw values. The results may be combined as e.g.

```
256L*as.integer(x@data[[i]]$msb) + as.integer(x@data[[i]]$lsb)
```

and this is what is returned by executing `x[["aerosol"]]`.

Landsat data files typically occupy approximately a gigabyte of storage. That means that corresponding Oce objects are about the same size, and this can pose significant problems on computers with less than 8GB of memory. It is sensible to specify bands of interest when reading data with `read.landsat`, and also to use `landsatTrim` to isolate geographical regions that need processing.

Experts may need to get direct access to the data, and this is easy because all Landsat objects (regardless of satellite) use a similar storage form. Band information is stored in byte form, to conserve space. Two bytes are used for each pixel in Landsat-8 objects, with just one for other objects. For example, if a Landsat-8 object named `L` contains the `tirs1` band, the most- and least-significant bytes will be stored in matrices `L@data$tirs1$msb` and `L@data$tirs1$lsb`. A similar Landsat-7 object would have the same items, but `msb` would be just the value `0x00`.

Derived bands, which may be added to a landsat object with `landsatAdd`, are not stored in byte matrices. Instead they are stored in numerical matrices, which means that they use 4X more storage space for Landsat-8 images, and 8X more storage space for other satellites. A computer needs at least 8GB of RAM to work with such data.

Landsat 8

The Landsat 8 satellite has 11 frequency bands, listed below (see [2]).

Band No.	Band Contents	Band Name	Wavelength (micrometers)	Resolution (meters)
1	Coastal aerosol	aerosol	0.43 - 0.45	30
2	Blue	blue	0.45 - 0.51	30
3	Green	green	0.53 - 0.59	30
4	Red	red	0.64 - 0.67	30
5	Near Infrared (NIR)	nir	0.85 - 0.88	30
6	SWIR 1	swir1	1.57 - 1.65	30
7	SWIR 2	swir2	2.11 - 2.29	30
8	Panchromatic	panchromatic	0.50 - 0.68	15
9	Cirrus	cirrus	1.36 - 1.38	30
10	Thermal Infrared (TIRS) 1	tirs1	10.60 - 11.19	100
11	Thermal Infrared (TIRS) 2	tirs2	11.50 - 12.51	100

In addition to the above, setting `band="terralook"` may be used as an abbreviation for `band=c("red", "green", "nir")`.

Band 8 is panchromatic, and has the highest resolution. For convenience of programming, `read.landsat` subsamples the `tirs1` and `tirs2` bands to the 30m resolution of the other bands. See Reference [3] for information about the evolution of Landsat 8 calibration coefficients, which as of summer 2014 are still subject to change.

Landsat 7

Band information is as follows (from [8]). The names are not official, but are set up to roughly correspond with Landsat-8 names, according to wavelength. An exception is the Landsat-7 bands named `tirs1` and `tirs2`, which are at two different gain settings, with identical wavelength span for each, which roughly matches the range of the Landsat-8 bands `tirs1` and `tirs2` combined. This may seem confusing, but it lets code like `plot(im, band="tirs1")` to work with both Landsat-8 and Landsat-7.

Band No.	Band Contents	Band Name	Wavelength (micrometers)	Resolution (meters)
1	Blue	blue	0.45 - 0.52	30
2	Green	green	0.52 - 0.60	30
3	Red	red	0.63 - 0.69	30
4	Near IR	nir	0.77 - 0.90	30
5	SWIR	swir1	1.55 - 1.75	30
6	Thermal IR	tirs1	10.4 - 12.50	30
7	Thermal IR	tirs2	10.4 - 12.50	30
8	SWIR	swir2	2.09 - 2.35	30
9	Panchromatic	panchromatic	0.52 - 0.90	15

Author(s)

Dan Kelley and Clark Richards

References

1. See the USGS "glovis" web site.
2. see landsat.gsfc.nasa.gov/?page_id=5377
3. see landsat.usgs.gov/calibration_notices.php
4. http://dankelley.github.io/r/2014/07/01/landsat.html
5. http://scienceofdoom.com/2010/12/27/emissivity-of-the-ocean/
6. see landsat.usgs.gov/Landsat8_Using_Product.php
7. see landsathandbook.gsfc.nasa.gov/pdfs/Landsat7_Handbook.pdf
8. see landsat.usgs.gov/band_designations_landsat_satellites.php

9. Yu, X. X. Guo and Z. Wu., 2014. Land Surface Temperature Retrieval from Landsat 8 TIRS-Comparison between Radiative Transfer Equation-Based Method, Split Window Algorithm and Single Channel Method, *Remote Sensing*, 6, 9829-9652. http://www.mdpi.com/2072-4292/6/10/9829

10. Rajeshwari, A., and N. D. Mani, 2014. Estimation of land surface temperature of Dindigul district using Landsat 8 data. *International Journal of Research in Engineering and Technology*, 3(5), 122-126. http://www.academia.edu/7655089/ESTIMATION_OF_LAND_SURFACE_TEMPERATURE_OF_DINDIGUL_DISTRICT_

11. Konda, M. Imasato N., Nishi, K., and T. Toda, 1994. Measurement of the Sea Surface Emissivity. *Journal of Oceanography*, 50, 17:30. http://www.terrapub.co.jp/journals/JO/pdf/5001/50010017.pdf

See Also

Data from AMSR satellites are handled with amsr-class.

A file containing Landsat data may be read with read.landsat or read.oce, and one such file is provided by the **ocedata** package as a dataset named landsat.

Plots may be made with plot,landsat-method. Since plotting can be quite slow, decimation is available both in the plotting function and as the separate function decimate. Images may be subsetted with landsatTrim.

landsat-class for handling data from the Landsat-8 satellite.

Other things related to landsat data: [[,landsat-method, landsatAdd, landsatTrim, landsat, plot,landsat-method, read.landsat, summary,landsat-method

Other things related to landsat data: [[,landsat-method, landsatAdd, landsatTrim, landsat, plot,landsat-method, read.landsat, summary,landsat-method

---

landsatAdd *Add a Band to a landsat Object*

Description

Add a band to an object of landsat-class. Note that it will be stored in numeric form, not raw form, and therefore it will require much more storage than data read with read.landsat.

Usage

```
landsatAdd(x, data, name, debug = getOption("oceDebug"))
```

Arguments

x	A landsat object, e.g. as read by read.landsat.
data	A matrix of data, with dimensions matching that of entries already in x.
name	The name to be used for the data, i.e. the data can later be accessed with d[[name]] where d is the name of the return value from the present function.
debug	A flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or a higher value for more debugging.

Value

An object of landsat-class, with a new data band.

Author(s)

Dan Kelley

See Also

The documentation for `landsat-class` explains the structure of landsat objects, and also outlines the other functions dealing with them.

Other things related to `landsat` data: `[[,landsat-method`, `landsat-class`, `landsatTrim`, `landsat`, `plot,landsat-method`, `read.landsat`, `summary,landsat-method`

---

landsatTrim — *Trim a landsat Image to a Geographical Region*

Description

Trim a landsat image to a latitude-longitude box. This is only an approximate operation, because landsat images are provided in x-y coordinates, not longitude-latitude coordinates.

Usage

```
landsatTrim(x, ll, ur, box, debug = getOption("oceDebug"))
```

Arguments

`x`	A `landsat` object, e.g. as read by `read.landsat`.
`ll`	A list containing `longitude` and `latitude`, for the lower-left corner of the portion of the image to retain, or a vector with first element longitude and second element latitude. If provided, then `ur` must also be provided, but `box` cannot.
`ur`	A list containing `longitude` and `latitude`, for the upper-right corner of the portion of the image to retain, or a vector with first element longitude and second element latitude. If provided, then `ll` must also be provided, but `box` cannot.
`box`	A list containing `x` and `y` (each of length 2), corresponding to the values for `ll` and `ur`, such as would be produced by a call to `locator(2)`. If provided, neither `ll` nor `ur` may be provided.
`debug`	A flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or a higher value for more debugging.

Details

As of June 25, 2015, the matrices storing the image data are trimmed to indices determined by linear interpolation based on the location of the `ll` and `ur` corners within the lon-lat corners specified in the image data. (A previous version trimmed in UTM space, and in fact this may be done in future also, if a problem in lonlat/utm conversion is resolved.) An error results if there is no intersection between the trimming box and the image box.

Value

An object of `landsat-class`, with data having been trimmed approximately as specified.

Author(s)

Dan Kelley and Clark Richards

See Also

The documentation for `landsat-class` explains the structure of landsat objects, and also outlines the other functions dealing with them.

Other things related to `landsat` data: `[[,landsat-method`, `landsat-class`, `landsatAdd`, `landsat`, `plot,landsat-method`, `read.landsat`, `summary,landsat-method`

`latFormat` *Format a latitude*

Description

Format a latitude, using "S" for negative latitude.

Usage

```
latFormat(lat, digits = max(6, getOption("digits") - 1))
```

Arguments

`lat`	latitude in °N north of the equator.
`digits`	the number of significant digits to use when printing.

Value

A character string.

Author(s)

Dan Kelley

See Also

`lonFormat` and `latlonFormat`.

latlonFormat *Format a latitude-longitude pair*

Description

Format a latitude-longitude pair, using "S" for negative latitudes, etc.

Usage

```
latlonFormat(lat, lon, digits = max(6, getOption("digits") - 1))
```

Arguments

lat	latitude in °N north of the equator.
lon	longitude in °N east of Greenwich.
digits	the number of significant digits to use when printing.

Value

A character string.

Author(s)

Dan Kelley

See Also

`latFormat` and `lonFormat`.

lisst *LISST Dataset*

Description

LISST (Laser in-situ scattering and transmissometry) dataset, constructed artificially.

Usage

```
data(lisst)
```

Author(s)

Dan Kelley

Source

This was constructed artificially using as.lisst,

See Also

Other datasets provided with oce: adp, adv, argo, cm, coastlineWorld, colors, ctdRaw, ctd, echosounder, landsat, lobo, met, rsk, sealevelTuktoyaktuk, sealevel, section, topoWorld, wind

---

lisst-class *Class to Store LISST Data*

Description

Class to store LISST (Laser in-situ scattering and transmissometry) data.

Details

One may read lisst objects with read.lisst, generate them with as.lisst, plot them with plot,lisst-method, and summarize them with summary,lisst-method. Elements may be extracted with [[,lisst-method or replaced with [[<-,lisst-method.

Author(s)

Dan Kelley

References

A users's manual for the LISST-100 instrument is available at the manufacturer's website http://www.sequoiasci.com.

See Also

Other classes provided by oce: adp-class, adv-class, argo-class, bremen-class, cm-class, coastline-class, ctd-class, echosounder-class, lobo-class, met-class, oce-class, odf-class, rsk-class, sealevel-class, section-class, topo-class, windrose-class

Other things related to lisst data: [[,lisst-method, [[<-,lisst-method, as.lisst, plot,lisst-method, read.lisst, summary,lisst-method lobo *LOBO Dataset*

Description

This is sample lobo dataset obtained in the Northwest Arm of Halifax by Satlantic.

Author(s)

Dan Kelley

Source

The data were downloaded from a web interface at Satlantic LOBO web server and then read with `read.lobo`.

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to lobo data: `[[,lobo-method`, `[[<-,lobo-method`, `as.lobo`, `lobo-class`, `plot,lobo-method`, `subset,lobo-method`, `summary,lobo-method`

Examples

```
## Not run:
library(oce)
data(lobo)
summary(lobo)
plot(lobo)

## End(Not run)
```

lobo-class *Class to Store LOBO Data*

Description

Class to store LOBO data. A `lobo` object may be read with `read.lobo` or constructed with `as.lobo`. Plots can be made with `plot,lobo-method`, while `summary,lobo-method` produces statistical summaries. Data within a `lobo` object may be retrieved with `[[,lobo-method` and altered with `[[,lobo-method`.

Author(s)

Dan Kelley

See Also

Other classes provided by oce: adp-class, adv-class, argo-class, bremen-class, cm-class, coastline-class, ctd-class, echosounder-class, lisst-class, met-class, oce-class, odf-class, rsk-class, sealevel-class, section-class, topo-class, windrose-class

Other things related to lobo data: [[,lobo-method, [[<-,lobo-method, as.lobo, lobo, plot,lobo-method, subset,lobo-method, summary,lobo-method

---

`lonFormat` *Format a longitude*

Description

Format a longitude, using "W" for west longitude.

Usage

```
lonFormat(lon, digits = max(6, getOption("digits") - 1))
```

Arguments

`lon`	longitude in °N east of Greenwich.
`digits`	the number of significant digits to use when printing.

Value

A character string.

Author(s)

Dan Kelley

See Also

latFormat and latlonFormat.

lonlat2map *Convert Longitude and Latitude to X and Y*

Description

If a projection is already being used (e.g. as set by mapPlot) then only longitude and latitude should be given, and the other arguments will be inferred by lonlat2map. This is important because otherwise, if a new projection is called for, it will ruin any additions to the existing plot.

Usage

```
lonlat2map(longitude, latitude, projection = "",
  debug = getOption("oceDebug"))
```

Arguments

longitude	a vector containing decimal longitudes, or a list containing items named longitude and latitude, in which case the indicated values are used, and next argument is ignored.
latitude	a vector containing decimal latitude (ignored if longitude is a list, as described above).
projection	optional indication of projection. This must be character string in the format used by the rgdal package; see mapPlot.)
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Value

A list containing x and y.

Author(s)

Dan Kelley

See Also

mapLongitudeLatitudeXY is a safer alternative, if a map has already been drawn with mapPlot, because that function cannot alter an existing projection. map2lonlat is an inverse to map2lonlat.

Other functions related to maps: lonlat2utm, map2lonlat, mapArrows, mapAxis, mapContour, mapDirectionField, mapGrid, mapImage, mapLines, mapLocator, mapLongitudeLatitudeXY, mapPlot, mapPoints, mapPolygon, mapScalebar, mapText, mapTissot, oceCRS, shiftLongitude, usrLonLat, utm2lonlat

Examples

```
## Not run:
library(oce)
## Cape Split, in the Minas Basin of the Bay of Fundy
cs <- list(longitude=-64.49657, latitude=45.33462)
xy <- lonlat2map(cs, projection="+proj=merc")
map2lonlat(xy)

## End(Not run)
```

`lonlat2utm` *Convert Longitude and Latitude to UTM*

Description

Convert Longitude and Latitude to UTM

Usage

```
lonlat2utm(longitude, latitude, zone, km = FALSE)
```

Arguments

`longitude`	decimal longitude. May also be a list containing items named `longitude` and `latitude`, in which case the indicated values are used, and next argument is ignored.
`latitude`	decimal latitude (ignored if `longitude` is a list containing both coordinates)
`zone`	optional indication of UTM zone. Normally this is inferred from the longitude, but specifying it can be helpful in dealing with Landsat images, which may cross zones and which therefore are described by a single zone.
`km`	logical value indicating whether `easting` and `northing` are in kilometers or meters.

Value

A list containing `easting`, `northing`, `zone` and `hemisphere`.

Author(s)

Dan Kelley

References

http://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system, downloaded May 31, 2014.

See Also

`utm2lonlat` does the inverse operation. For general projections and their inverses, use `lonlat2map` and `map2lonlat`.

Other functions related to maps: `lonlat2map`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
library(oce)
## Cape Split, in the Minas Basin of the Bay of Fundy
lonlat2utm(-64.496567, 45.334626)
```

`lookWithin` *Look Within the First Element of a List for Replacement Values*

Description

Look Within the First Element of a List for Replacement Values

Usage

```
lookWithin(list)
```

Arguments

`list`	A list of elements, typically arguments that will be used in sw functions.

Details

This is a helper function used by various seawater functions. It is used for a call like `swRho(ctd)`, in which the first argument, which is normally `salinity` may be an object that contains salinity plus the other items that `swRho` expects to see as arguments. This shorthand is very helpful in calls to the suite of `sw` functions. If this first argument is an object of this sort, then the other arguments are ignored *except* for two special cases:

- an item named `eos` is copied directly from `list`
- if the object stores `temperature` defined with the IPTS-68 scale, then `T90fromT68` is used to convert to the ITS-90 scale, because this is what is expected in most seawater functions. (For example, the RMS difference between these temperature variants is 0.002C for the `ctd` dataset.)

Value

A list with elements of the same names but possibly filled in from the first element.

lowpass *Perform lowpass digital filtering*

Description

The filter coefficients are constructed using standard definitions, and then filter in the **stats** package is used to filter the data. This leaves NA values within half the filter length of the ends of the time series, but these may be replaced with the original x values, if the argument replace is set to TRUE.

Usage

```
lowpass(x, filter = "hamming", n, replace = TRUE, coefficients = FALSE)
```

Arguments

x	a vector to be smoothed
filter	name of filter; at present, "hamming", "hanning", and "boxcar" are permitted.
n	length of filter (must be an odd integer exceeding 1)
replace	a logical value indicating whether points near the ends of x should be copied into the end regions, replacing the NA values that would otherwise be placed there by filter.
coefficients	logical value indicating whether to return the filter coefficients, instead of the filtered values. In accordance with conventions in the literature, the returned values are not normalized to sum to 1, although of course that normalization is done in the actual filtering.

Value

By default, lowpass returns a filtered version of x, but if coefficients is TRUE then it returns the filter coefficients.

Caution

This function was added in June of 2017, and it may be extended during the rest of 2017. New arguments may appear between n and replace, so users are advised to call this function with named arguments, not positional arguments.

Author(s)

Dan Kelley

Examples

```
library(oce)
par(mfrow=c(1, 2), mar=c(4, 4, 1, 1))
coef <- lowpass(n=5, coefficients=TRUE)
plot(-2:2, coef, ylim=c(0, 1), xlab="Lag", ylab="Coefficient")
x <- seq(-5, 5) + rnorm(11)
plot(1:11, x, type='o', xlab="time", ylab="x and X")
X <- lowpass(x, n=5)
lines(1:11, X, col=2)
points(1:11, X, col=2)
```

`magneticField` *Earth magnetic declination, inclination, and intensity*

Description

Implements the 12th generation International Geomagnetic Reference Field (IGRF), based on a reworked version of a Fortran program downloaded from a NOAA website [1]. The code (subroutine `igrf12syn`) seems to have been written by Susan Macmillan of the British Geological Survey. Comments in the code indicate that it employs coefficients agreed to in December 2014 by the IAGA Working Group V-MOD. Comments in the `igrf12syn` source code also suggest that the valid time interval is from years 1900 to 2020, with only the values from 1945 to 2010 being considered definitive.

Usage

```
magneticField(longitude, latitude, time)
```

Arguments

`longitude`	longitude in degrees east (negative for degrees west). The dimensions must conform to lat.
`latitude`	latitude in degrees north, a number, vector, or matrix.
`time`	either a decimal year or a POSIX time corresponding to the `longitude` and `latitude` values, or a vector or matrix matching these location values.

Value

A list containing `declination`, `inclination`, and `intensity`.

Author(s)

Dan Kelley wrote the R code and a fortran wrapper to the `igrf12.f` subroutine, which was written by Susan Macmillan of the British Geological Survey and distributed “without limitation” (email from SM to DK dated June 5, 2015).

References

1. The underlying Fortran code is from igrf12.f, downloaded the NOAA website ('https://www.ngdc.noaa.gov/IAGA/vm on June 7, 2015. (That website suggests that there have been no update to the algorithm as of March 29, 2017.)

See Also

Other things related to magnetism: applyMagneticDeclination

Examples

```
library(oce)
# Halifax NS
magneticField(-(63+36/60), 44+39/60, 2013)

## Not run:
## map of North American values
data(coastlineWorld)
mapPlot(coastlineWorld, longitudelim=c(-130,-55), latitudelim=c(35,60),
        projection="+proj=lcc +lat_0=20 +lat_1=60 +lon_0=-100")
lon <- seq(-180, 180, 1)
lat <- seq(-90, 90)
lonm <- rep(lon, each=length(lat))
latm <- rep(lat, times=length(lon))
## Note the counter-intuitive nrow argument
decl <- matrix(magneticField(lonm, latm, 2013)$declination,
               nrow=length(lon), byrow=TRUE)
mapContour(lon, lat, decl, col='red', levels=seq(-90, 90, 5))
incl <- matrix(magneticField(lonm, latm, 2013)$inclination,
               nrow=length(lon), byrow=TRUE)
mapContour(lon, lat, incl, col='blue', levels=seq(-90, 90, 5))

## End(Not run)
```

---

makeFilter *Make a digital filter*

Description

The filter is suitable for use by filter, convolve or (for the asKernal=TRUE case) with kernapply. Note that convolve should be faster than filter, but it cannot be used if the time series has missing values. For the Blackman-Harris filter, the half-power frequency is at 1/m cycles per time unit, as shown in the "Examples" section. When using filter or kernapply with these filters, use circular=TRUE.

Usage

```
makeFilter(type = c("blackman-harris", "rectangular", "hamming", "hann"), m,
  asKernel = TRUE)
```

Arguments

`type` a string indicating the type of filter to use. (See Harris (1978) for a comparison of these and similar filters.)

- `"blackman-harris"` yields a modified raised-cosine filter designated as "4-Term (-92 dB) Blackman-Harris" by Harris (1978; coefficients given in the table on page 65). This is also called "minimum 4-sample Blackman Harris" by that author, in his Table 1, which lists figures of merit as follows: highest side lobe level -92dB; side lobe fall off -6 db/octave; coherent gain 0.36; equivalent noise bandwidth 2.00 bins; 3.0-dB bandwidth 1.90 bins; scallop loss 0.83 dB; worst case process loss 3.85 dB; 6.0-db bandwidth 2.72 bins; overlap correlation 46 percent for 75% overlap and 3.8 for 50% overlap. Note that the equivalent noise bandwidth is the width of a spectral peak, so that a value of 2 indicates a cutoff frequency of `1/m`, where `m` is as given below.
- `"rectangular"` for a flat filter. (This is just for convenience. Note that `kernel("daniell",....)` gives the same result, in kernel form.) `"hamming"` for a Hamming filter (a raised-cosine that does not taper to zero at the ends)
- `"hann"` (a raised cosine that tapers to zero at the ends).

`m` length of filter. This should be an odd number, for any non-rectangular filter.

`asKernel` boolean, set to `TRUE` to get a smoothing kernel for the return value.

Value

If `asKernel` is `FALSE`, this returns a list of filter coefficients, symmetric about the midpoint and summing to 1. These may be used with `filter`, which should be provided with argument `circular=TRUE` to avoid phase offsets. If `asKernel` is `TRUE`, the return value is a smoothing kernel, which can be applied to a timeseries with `kernapply`, whose bandwidth can be determined with `bandwidth.kernel`, and which has both print and plot methods.

Author(s)

Dan Kelley

References

F. J. Harris, 1978. On the use of windows for harmonic analysis with the discrete Fourier Transform. *Proceedings of the IEEE*, 66(1), 51-83 (`http://web.mit.edu/xiphmont/Public/windows.pdf`.)

Examples

```
library(oce)

# 1. Demonstrate step-function response
y <- c(rep(1, 10), rep(-1, 10))
x <- seq_along(y)
plot(x, y, type='o', ylim=c(-1.05, 1.05))
BH <- makeFilter("blackman-harris", 11, asKernel=FALSE)
```

```
H <- makeFilter("hamming", 11, asKernel=FALSE)
yBH <- stats::filter(y, BH)
points(x, yBH, col=2, type='o')
yH <- stats::filter(y, H)
points(yH, col=3, type='o')
legend("topright", col=1:3, cex=2/3, pch=1,
       legend=c("input", "Blackman Harris", "Hamming"))

# 2. Show theoretical and practical filter gain, where
#    the latter is based on random white noise, and
#    includes a particular value for the spans
#    argument of spectrum(), etc.
## Not run:  # need signal package for this example
r <- rnorm(2048)
rh <- stats::filter(r, H)
rh <- rh[is.finite(rh)] # kludge to remove NA at start/end
sR <- spectrum(r, plot=FALSE, spans=c(11, 5, 3))
sRH <- spectrum(rh, plot=FALSE, spans=c(11, 5, 3))
par(mfrow=c(2, 1), mar=c(3, 3, 1, 1), mgp=c(2, 0.7, 0))
plot(sR$freq, sRH$spec/sR$spec, xlab="Frequency", ylab="Power Transfer",
     type='l', lwd=5, col='gray')
theory <- freqz(H, n=seq(0,pi,length.out=100))
# Note we must square the modulus for the power spectrum
lines(theory$f/pi/2, Mod(theory$h)^2, lwd=1, col='red')
grid()
legend("topright", col=c("gray", "red"), lwd=c(5, 1), cex=2/3,
       legend=c("Practical", "Theory"), bg="white")
plot(log10(sR$freq), log10(sRH$spec/sR$spec),
     xlab="log10 Frequency", ylab="log10 Power Transfer",
     type='l', lwd=5, col='gray')
theory <- freqz(H, n=seq(0,pi,length.out=100))
# Note we must square the modulus for the power spectrum
lines(log10(theory$f/pi/2), log10(Mod(theory$h)^2), lwd=1, col='red')
grid()
legend("topright", col=c("gray", "red"), lwd=c(5, 1), cex=2/3,
       legend=c("Practical", "Theory"), bg="white")

## End(Not run)
```

`makeSection` *Make a Section (DEFUNCT)*

Description

This is a defunct function; use `as.section` instead, and see oce-defunct for more on the oce procedure for retiring functions.

Usage

```
makeSection(item, ...)
```

Arguments

`item`	Ignored, since this function is defunct
`...`	Ignored, since this function is defunct

`map2lonlat` *Convert X and Y to Longitude and Latitude*

Description

Convert from x-y coordinates to longitude and latitude. This is normally called internally within oce; see 'Bugs'.

Usage

```
map2lonlat(x, y, init = c(0, 0))
```

Arguments

`x`	vector containing the x component of points in the projected space, or a list containing items named `x` and `y`, in which case the next argument is ignored.
`y`	vector containing the y coordinate of points in the projected space (ignored if `x` is a list, as described above).
`init`	vector containing the initial guesses for longitude and latitude, presently ignored.

Details

A projection must already have been set up, by a call to `mapPlot` or `lonlat2map`. It should be noted that not all projections are handled well; see 'Bugs'.

Value

A list containing `longitude` and `latitude`, with `NA` values indicating points that are off the globe as displayed.

Bugs

`oce` uses project in the **rgdal** package to handle projections. Only those projections that have inverses are permitted within `oce`, and even those can sometimes yield errors, owing to limitations in **rgdal**.

Author(s)

Dan Kelley

See Also

lonlat2map does the inverse operation.

A map must first have been created with mapPlot.

Other functions related to maps: lonlat2map, lonlat2utm, mapArrows, mapAxis, mapContour, mapDirectionField, mapGrid, mapImage, mapLines, mapLocator, mapLongitudeLatitudeXY, mapPlot, mapPoints, mapPolygon, mapScalebar, mapText, mapTissot, oceCRS, shiftLongitude, usrLonLat, utm2lonlat

Examples

```
## Not run:
library(oce)
## Cape Split, in the Minas Basin of the Bay of Fundy
cs <- list(longitude=-64.49657, latitude=45.33462)
xy <- lonlat2map(cs, projection="+proj=merc")
map2lonlat(xy)

## End(Not run)
```

---

mapArrows *Add Arrows to a Map*

Description

Plot arrows on an existing map, e.g. to indicate a place location. This is not well-suited for drawing direction fields, e.g. of velocities; for that, see mapDirectionField.

Usage

```
mapArrows(longitude0, latitude0, longitude1 = longitude0,
  latitude1 = latitude0, length = 0.25, angle = 30, code = 2,
  col = par("fg"), lty = par("lty"), lwd = par("lwd"), ...)
```

Arguments

`longitude0, latitude0`	starting points for arrows.
`longitude1, latitude1`	ending points for arrows.
`length`	length of the arrow heads, passed to arrows.
`angle`	angle of the arrow heads, passed to arrows.
`code`	numerical code indicating the type of arrows, passed to arrows.
`col`	arrow colour, passed to arrows.
`lty`	arrow line type, passed to arrows.
`lwd`	arrow line width, passed to arrows.
`...`	optional arguments passed to arrows.

Details

Adds arrows to an existing map, by analogy to arrows.

Author(s)

Dan Kelley

See Also

A map must first have been created with mapPlot.

Other functions related to maps: lonlat2map, lonlat2utm, map2lonlat, mapAxis, mapContour, mapDirectionField, mapGrid, mapImage, mapLines, mapLocator, mapLongitudeLatitudeXY, mapPlot, mapPoints, mapPolygon, mapScalebar, mapText, mapTissot, oceCRS, shiftLongitude, usrLonLat, utm2lonlat

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, longitudelim=c(-120, -60), latitudelim=c(30, 60),
        col="lightgray", projection="+proj=lcc +lon_0=-100")
lon <- seq(-120, -75, 15)
n <- length(lon)
lat <- 45 + rep(0, n)
# Draw meridional arrows in N America, from 45N to 60N.
mapArrows(lon, lat, lon, lat+15, length=0.05, col="blue")

## End(Not run)
```

---

mapAxis *Add Axis Labels to an Existing Map*

Description

Plot axis labels on an existing map.

Usage

```
mapAxis(side = 1:2, longitude = NULL, latitude = NULL, tick = TRUE,
  line = NA, pos = NA, outer = FALSE, font = NA, lty = "solid",
  lwd = 1, lwd.ticks = lwd, col = NULL, col.ticks = NULL, hadj = NA,
  padj = NA, tcl = -0.3, cex.axis = 1, mgp = c(0, 0.5, 0),
  debug = getOption("oceDebug"))
```

Arguments

`side`	the side at which labels are to be drawn. If not provided, sides 1 and 2 will be used (i.e. bottom and left-hand sides).
`longitude`	vector of longitudes to indicate. If not provided, and if a grid has already been drawn, then the labels will be at the intersections of the grid lines with the plotting box.
`latitude`	vector of latitudes to indicate. If not provided, and if a grid has already been drawn, then the labels will be at the interesections of the grid lines with the plotting box.
`tick`	parameter passed to `axis`.
`line`	parameter passed to `axis`.
`pos`	parameter passed to `axis`.
`outer`	parameter passed to `axis`.
`font`	axis font, passed to `axis`.
`lty`	axis line type, passed to `axis`.
`lwd`	axis line width, passed to `axis`).
`lwd.ticks`	tick line width, passed to `axis`.
`col`	axis colour, passed to `axis`.
`col.ticks`	axis tick colour, passed to `axis`.
`hadj`	an argument that is transmitted to `axis`.
`padj`	an argument that is transmitted to `axis`.
`tcl`	axis-tick size (see `par`).
`cex.axis`	axis-label expansion factor (see `par`).
`mgp`	three-element numerical vector describing axis-label placement (see `par`). It usually makes sense to set the first and third elements to zero.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

This function is still in development, and the argument list as well as the action taken are both subject to change, hence the brevity of this help page.

Note that if a grid line crosses the axis twice, only one label will be drawn.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
par(mar=c(2, 2, 3, 1))
lonlim <- c(-180, 180)
latlim <- c(60, 120)
mapPlot(coastlineWorld, projection="+proj=stere +lat_0=90",
        longitudelim=lonlim, latitudelim=latlim,
        grid=FALSE)
mapGrid(15, 15, polarCircle=1/2)
mapAxis()

## End(Not run)
```

---

mapContour *Add Contours on a Existing map*

Description

Plot contours on an existing map.

Usage

```
mapContour(longitude = seq(0, 1, length.out = nrow(z)), latitude = seq(0, 1,
  length.out = ncol(z)), z, nlevels = 10, levels = pretty(range(z, na.rm =
  TRUE), nlevels), col = par("fg"), lty = par("lty"), lwd = par("lwd"))
```

Arguments

`longitude`	vector of longitudes of points to be plotted, or an object of class `topo` (see `topo-class`), in which case `longitude`, `latitude` and `z` are inferred from that object.
`latitude`	vector of latitudes of points to be plotted.
`z`	matrix to be contoured.
`nlevels`	number of contour levels, if and only if `levels` is not supplied.
`levels`	vector of contour levels.
`col`	line colour.
`lty`	line type.
`lwd`	line width.

Details

Adds contour lines to an existing map, using `mapLines`. The arguments are based on those to `contour` and `contourLines`.

Bugs

As with mapLines, long lines should be subdivided into multiple segments so that e.g. great circle lines will be curved.

Author(s)

Dan Kelley

See Also

A map must first have been created with mapPlot.

Other functions related to maps: lonlat2map, lonlat2utm, map2lonlat, mapArrows, mapAxis, mapDirectionField, mapGrid, mapImage, mapLines, mapLocator, mapLongitudeLatitudeXY, mapPlot, mapPoints, mapPolygon, mapScalebar, mapText, mapTissot, oceCRS, shiftLongitude, usrLonLat, utm2lonlat

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
par(mar=rep(1, 4))
## Arctic 100m, 2km, 3km isobaths, showing shelves and ridges.
mapPlot(coastlineWorld, latitudelim=c(60, 120), longitudelim=c(-130,-50),
        projection="+proj=stere +lat_0=90")
data(topoWorld)
lon <- topoWorld[['longitude']]
lat <- topoWorld[['latitude']]
z <- topoWorld[['z']]
mapContour(lon, lat, z, levels=c(-100, -2000, -3000), col=1:3, lwd=2)

## End(Not run)
```

---

mapCoordinateSystem *Draw a coordinate system*

Description

Draws arrows on a map to indicate a coordinate system, e.g. for an to indicate a coordinate system set up so that one axis is parallel to a coastline.

Usage

```
mapCoordinateSystem(longitude, latitude, L = 100, phi = 0, ...)
```

Arguments

longitude	numeric value of longitude in degrees.
latitude	numeric value of latitude in degrees.
L	axis length in km.
phi	angle, in degrees counterclockwise, that the "x" axis makes to a line of latitude.
...	plotting arguments, passed to mapArrows; see "Examples" for how to control the arrow-head size.

Details

This is a preliminary version of this function. It only works if the lines of constant latitude are horizontal on the plot.

Author(s)

Chantelle Layton

Examples

```
## Not run:
library(oce)
data(coastlineWorldFine, package='ocedata')
HfxLon <- -63.5752
HfxLat <- 44.6488
mapPlot(coastlineWorldFine, proj='+proj=merc',
        longitudelim=HfxLon+c(-2,2), latitudelim=HfxLat+c(-2,2),
        col='lightgrey')
mapCoordinateSystem(HfxLon, HfxLat, phi=45, length=0.05)

## End(Not run)
```

---

mapDirectionField   *Add a Direction Field to an Existing Map*

Description

Plot a direction field on a existing map.

Usage

```
mapDirectionField(longitude, latitude, u, v, scale = 1, length = 0.05,
  code = 2, col = par("fg"), ...)
```

Arguments

`longitude, latitude`
: vectors of the starting points for arrows.

`u, v`
: components of a vector to be shown as a direction field.

`scale`
: latitude degrees per unit of `u` or `v`.

`length`
: length of arrow heads, passed to `arrows`.

`code`
: code of arrows, passed to `arrows`.

`col`
: colour of arrows. This may be a single colour, or a matrix of colours of the same dimension as `u`.

`...`
: optional arguments passed to `arrows`, e.g. `angle` and `lwd` can be useful in differentiating different fields.

Details

Adds arrows for a direction field on an existing map. There are different possibilities for how `longitude`, `latitude` and `u` and `v` match up. In one common case, all four of these are matrices, e.g. output from a numerical model. In another, `longitude` and `latitude` are the coordinates along the matrices, and are thus stored in vectors with lengths that match appropriately.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
par(mar=rep(2, 4))
mapPlot(coastlineWorld, longitudelim=c(-120,-55), latitudelim=c(35, 50),
        proj="+proj=laea +lat0=40 +lat1=60" +lon_0=-110)
lon <- seq(-120, -60, 15)
lat <- 45 + seq(-15, 15, 5)
lonm <- matrix(expand.grid(lon, lat)[, 1], nrow=length(lon))
latm <- matrix(expand.grid(lon, lat)[, 2], nrow=length(lon))
## vectors pointed 45 degrees clockwise from north
u <- matrix(1/sqrt(2), nrow=length(lon), ncol=length(lat))
v <- matrix(1/sqrt(2), nrow=length(lon), ncol=length(lat))
mapDirectionField(lon, lat, u, v, scale=3)
mapDirectionField(lonm, latm, 0, 1, scale=3, col='red')
```

```
# Color code by longitude, using thick lines
col <- colormap(lonm)$zcol
mapDirectionField(lonm, latm, 1, 0, scale=3, col=col, lwd=2)

## End(Not run)
```

`mapGrid` *Add a Longitude and Latitude Grid to a Map*

Description

Plot longitude and latitude grid on an existing map.

Usage

```
mapGrid(dlongitude = 15, dlatitude = 15, longitude, latitude,
  col = "darkgray", lty = "solid", lwd = 0.5 * par("lwd"),
  polarCircle = 0, longitudelim, latitudelim, debug = getOption("oceDebug"))
```

Arguments

`dlongitude`	increment in longitude, ignored if `longitude` is supplied, but otherwise determines the longitude sequence.
`dlatitude`	increment in latitude, ignored if `latitude` is supplied, but otherwise determines the latitude sequence.
`longitude`	vector of longitudes, or `NULL` to prevent drawing longitude lines.
`latitude`	vector of latitudes, or `NULL` to prevent drawing latitude lines.
`col`	colour of lines
`lty`	line type
`lwd`	line width
`polarCircle`	a number indicating the number of degrees of latitude extending from the poles, within which zones are not drawn.
`longitudelim`	optional argument specifying suggested longitude limits for the grid. If this is not supplied, grid lines are drawn for the whole globe, which can yield excessively slow drawing speeds for small-region plots. This, and `latitudelim`, are both set by `mapPlot` if the arguments of the same name are passed to that function.
`latitudelim`	similar to `longitudelim`.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

This is somewhat analogous to `grid`, except that the first two arguments of the latter supply the number of lines in the grid, whereas the present function has increments for the first two arguments.

Plans

At the moment, the function cannot determine which lines might work with labels on axes, but this could perhaps be added later, making this more analogous with `grid`.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, type='l', grid=FALSE,
longitudelim=c(-80, 10), latitudelim=c(0, 120),
projection="+proj=ortho")
mapGrid(15, 15, polarCircle=15)

## End(Not run)
```

`mapImage` *Add an Image to a Map*

Description

Plot an image on an existing map that was created with `mapPlot`. (See example 4 for a way to start with a blank map.)

Usage

```
mapImage(longitude, latitude, z, zlim, zclip = FALSE, breaks, col, colormap,
  border = NA, lwd = par("lwd"), lty = par("lty"), missingColor = NA,
  filledContour = FALSE, gridder = "binMean2D",
  debug = getOption("oceDebug"))
```

Arguments

`longitude`	vector of longitudes corresponding to `z` matrix.
`latitude`	vector of latitudes corresponding to `z` matrix.
`z`	matrix to be represented as an image.
`zlim`	limit for z (colour).
`zclip`	A logical value, `TRUE` indicating that out-of-range `z` values should be painted with `missingColor` and `FALSE` indicating that these values should be painted with the nearest in-range colour. If `zlim` is given then its min and max set the range. If `zlim` is not given but `breaks` is given, then the min and max of `breaks` sets the range used for z. If neither `zlim` nor `breaks` is given, clipping is not done, i.e. the action is as if `zclip` were `FALSE`.
`breaks`	The z values for breaks in the colour scheme. If this is of length 1, the value indicates the desired number of breaks, which is supplied to `pretty`, in determining clean break points.
`col`	Either a vector of colours corresponding to the breaks, of length 1 plus the number of breaks, or a function specifying colours, e.g. `oce.colorsJet` for a rainbow.
`colormap`	optional colormap, as created by `colormap`. If a `colormap` is provided, then its properties takes precedence over `breaks`, `col`, `missingColor`, and `zclip` specified to `mapImage`.
`border`	Colour used for borders of patches (passed to `polygon`); the default `NA` means no border.
`lwd`	line width, used if borders are drawn.
`lty`	line type, used if borders are drawn.
`missingColor`	a colour to be used to indicate missing data, or `NA` to skip the drawing of such regions (which will retain whatever material has already been drawn at the regions).
`filledContour`	either a logical value indicating whether to use filled contours to plot the image, or a numerical value indicating the resampling rate to be used in interpolating from lon-lat coordinates to x-y coordinates. See “Details” for how this interacts with `gridder`.
`gridder`	Name of gridding function used if `filledContour` is `TRUE`. This can be either `"binMean2D"` to select `binMean2D` or `"interp"` for `interp`. If not provided, then a selection is made automatically, with `binMean2D` being used if there are more than 10,000 data points in the present graphical view. This `"binMean2D"` method is much faster than `"interp"`.
`debug`	A flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

Adds an image to an existing map, by analogy to `image`.

The data are on a regular grid in lon-lat space, but not in the projected x-y space. This means that `image` cannot be used. Instead, there are two approaches, depending on the value of `filledContour`.

If filledContour is FALSE, the image "pixels" are with polygon, which can be prohibitively slow for fine grids. However, if filledContour is TRUE or a numerical value, then the "pixels" are remapped into a regular grid and then displayed with .filled.contour. The remapping starts by converting the regular lon-lat grid to an irregular x-y grid using lonlat2map. This irregular grid is then interpolated onto a regular x-y grid with binMean2D or with interp from the akima package, as determined by the gridder argument. If filledContour is TRUE, the dimensions of the regular x-y grid is the same as that of the original lon-lat grid; otherwise, the number of rows and columns are multiplied by the numerical value of filledContour, e.g. the value 2 means to make the grid twice as fine.

Filling contours can produce aesthetically-pleasing results, but the method involves interpolation, so the data are not represented exactly and analysts are advised to compare the results from the two methods (and perhaps various grid refinement values) to guard against misinterpretation.

If a png device is to be used, it is advised to supply arguments type="cairo" and antialias="none"; see [1].

Author(s)

Dan Kelley

References

1. http://codedocean.wordpress.com/2014/02/03/anti-aliasing-and-image-plots/

See Also

A map must first have been created with mapPlot.

Other functions related to maps: lonlat2map, lonlat2utm, map2lonlat, mapArrows, mapAxis, mapContour, mapDirectionField, mapGrid, mapLines, mapLocator, mapLongitudeLatitudeXY, mapPlot, mapPoints, mapPolygon, mapScalebar, mapText, mapTissot, oceCRS, shiftLongitude, usrLonLat, utm2lonlat

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
data(topoWorld)

## 1. topography
par(mfrow=c(2, 1), mar=c(2, 2, 1, 1))
lonlim <- c(-70, -50)
latlim <- c(40, 50)
topo <- decimate(topoWorld, by=2) # coarse to illustrate filled contours
topo <- subset(topo, latlim[1] < latitude & latitude < latlim[2])
topo <- subset(topo, lonlim[1] < longitude & longitude < lonlim[2])
mapPlot(coastlineWorld, type='l',
        longitudelim=lonlim, latitudelim=latlim,
        projection="+proj=lcc +lat_1=40 +lat_2=50 +lon_0=-60")
breaks <- seq(-5000, 1000, 500)
mapImage(topo, col=oce.colorsGebco, breaks=breaks)
```

```
mapLines(coastlineWorld)
box()
mapPlot(coastlineWorld, type='l',
        longitudelim=lonlim, latitudelim=latlim,
        projection="+proj=lcc +lat_1=40 +lat_2=50 +lon_0=-60")
mapImage(topo, filledContour=TRUE, col=oce.colorsGebco, breaks=breaks)
box()
mapLines(coastlineWorld)

## 2. Northern polar region, with colour-coded bathymetry
par(mfrow=c(1,1))
drawPalette(c(-5000, 0), zlim=c(-5000, 0), col=oce.colorsJet)
mapPlot(coastlineWorld, projection="+proj=stere +lat_0=90",
        longitudelim=c(-180,180), latitudelim=c(60,120))
mapImage(topoWorld, zlim=c(-5000, 0), col=oce.colorsJet)
mapLines(coastlineWorld[['longitude']], coastlineWorld[['latitude']])

## 3. Levitus SST
par(mfrow=c(1,1))
data(levitus, package='ocedata')
lon <- levitus$longitude
lat <- levitus$latitude
SST <- levitus$SST
par(mar=rep(1, 4))
Tlim <- c(-2, 30)
drawPalette(Tlim, col=oce.colorsJet)
mapPlot(coastlineWorld, projection="+proj=moll", grid=FALSE)
mapImage(lon, lat, SST, col=oce.colorsJet, zlim=Tlim)
mapPolygon(coastlineWorld, col='gray')

## 4. Topography without drawing a coastline first
data(topoWorld)
cm <- colormap(topoWorld[['z']], name='gmt_relief')
drawPalette(colormap=cm)
mapPlot(c(-180,180), c(-90,90), type="n") # defaults to moll projection
mapImage(topoWorld, colormap=cm)

## End(Not run)
```

---

mapLines *Add Lines to a Map*

Description

Plot lines on an existing map

Usage

```
mapLines(longitude, latitude, greatCircle = FALSE, ...)
```

Arguments

`longitude`	vector of longitudes of points to be plotted, or an object from which longitude and latitude can be inferred (e.g. a coastline file, or the return value from `mapLocator`), in which case the following two arguments are ignored.
`latitude`	vector of latitudes of points to be plotted.
`greatCircle`	a logical value indicating whether to render line segments as great circles. (Ignored.)
`...`	optional arguments passed to `lines`.

Details

Adds lines to an existing map, by analogy to `lines`.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, type='l',
        longitudelim=c(-80, 10), latitudelim=c(0, 120),
        projection="+proj=ortho +lon_0=-40")
lon <- c(-63.5744, 0.1062)            # Halifax CA to London UK
lat <- c(44.6479, 51.5171)
mapPoints(lon, lat, col='red')
mapLines(lon, lat, col='red')

## End(Not run)
```

mapLocator — *Locate Points on a Map*

Description

Locate points on an existing map.

Usage

```
mapLocator(n = 512, type = "n", ...)
```

Arguments

`n`	number of points to locate; see `locator`.
`type`	type of connector for the points; see `locator`.
`...`	extra arguments passed to `locator` (and either `mapPoints` or `mapLines`, if appropriate) if `type` is not `'n'`.

Details

This uses `map2lonlat` to infer the location in geographical space; see the documentation for that function on its limitations.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

mapLongitudeLatitudeXY — *Convert From Longitude and Latitude to X and Y*

Description

Find (x, y) values corresponding to (longitude, latitude) values, using the present projection.

Usage

```
mapLongitudeLatitudeXY(longitude, latitude)
```

Arguments

`longitude` vector of the longitudes of points, or an object from which both latitude and longitude can be inferred (e.g. a coastline file, or the return value from `mapLocator`), in which case the following two arguments are ignored.

`latitude` vector of latitudes of points, needed only if they cannot be inferred from the first argument.

Details

This is mainly a wrapper around `lonlat2map`.

Value

A list containing `x` and `y`.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
par(mfrow=c(2, 1), mar=rep(2, 4))
mapPlot(coastlineWorld, projection="+proj=moll") # sets a projection
xy <- mapLongitudeLatitudeXY(coastlineWorld)
plot(xy, type='l', asp=1)

## End(Not run)
```

`mapMeridians` *Add Meridians on a Map [deprecated]*

Description

WARNING: This function will be removed soon; see oce-deprecated. Use `mapGrid` instead of the present function.

Usage

```
mapMeridians(latitude, lty = "solid", lwd = 0.5 * par("lwd"),
  col = "darkgray", ...)
```

Arguments

`latitude`	either a logical value indicating whether to draw a meridian grid, or a vector of latitudes at which to draw meridians.
`lty`	line type.
`lwd`	line width.
`col`	line colour.
`...`	optional arguments passed to `lines`.

Details

Plot meridians (lines of constant latitude) on an existing map.

This function should not be used, since it will be removed soon. Please use mapGrid() instead.

Author(s)

Dan Kelley

See Also

Other functions that will be removed soon: `addColumn`, `ctdAddColumn`, `ctdUpdateHeader`, `mapZones`, `oce.as.POSIXlt`

`mapPlot` *Draw a Map*

Description

Plot coordinates as a map, using one of the subset of projections provided by the **rgdal** package. The projection information specified with the `mapPlot` call is stored so that can be retrieved by related functions, making it easy to add more items so the map, including points, lines, text, images and contours.

Usage

```
mapPlot(longitude, latitude, longitudelim, latitudelim, grid = TRUE, bg, fill,
  border = NULL, col = NULL, clip = TRUE, type = "polygon",
  axes = TRUE, cex, cex.axis = 1, mgp = c(0, 0.5, 0), drawBox = TRUE,
  showHemi = TRUE, polarCircle = 0, lonlabel = NULL, latlabel = NULL,
  sides = NULL, projection = "+proj=moll", tissot = FALSE, trim = TRUE,
  debug = getOption("oceDebug"), ...)
```

Arguments

`longitude` either a vector of longitudes of points to be plotted, or something (an `oce` object, a list, or a data frame) from which both longitude and latitude may be inferred (in which case the `latitude` argument is ignored). If `longitude` is missing, both it and `latitude` are taken from `coastlineWorld`.

`latitude` vector of latitudes of points to be plotted (ignored if the first argument contains both latitude and longitude).

`longitudelim` optional vector of length two, indicating the longitude limits of the plot. This value is used in the selection of longitude lines that are shown (and possibly labelled on the axes). In some cases, e.g. for polar views, this can lead to odd results, with some expected longitude lines being left out of the plot. Altering `longitudelim` can often help in such cases, e.g. `longitudelim=c(-180, 180)` will force the drawing of lines all around the globe.

`latitudelim` optional vector of length two, indicating the latitude limits of the plot. This, together with `longitudelim` (and, importantly, the geometry of the plot device) is used in the selection of map scale.

`grid` either a number (or pair of numbers) indicating the spacing of longitude and latitude lines, in degrees, or a logical value (or pair of values) indicating whether to draw an auto-scaled grid, or whether to skip the grid drawing. In the case of numerical values, `NA` can be used to turn off the grid in longitude or latitude. Grids are set up based on examination of the scale used in middle 10 percent of the plot area, and for most projections this works quite well. If not, one may set `grid=FALSE` and add a grid later with `mapGrid`.

`bg` colour of the background (ignored).

`fill` **(deprecated)** is a deprecated argument; see oce-deprecated.

`border` colour of coastlines and international borders (ignored unless `type="polygon"`.

`col` either the colour for filling polygons (if `type="polygon"`) or the colour of the points and line segments (if `type="p"`, `type="l"`, or `type="o"`). If `col=NULL` then a default will be set: no coastline filling for the `type="polygon"` case, or black coastlines, for `type="p"`, `type="l"`, or `type="o"`.

`clip` logical value indicating whether to trim any coastline elements that lie wholly outside the plot region. This can prevent e.g. a problem of filling the whole plot area of an Arctic stereopolar view, because the projected trace for Antarctica lies outside all other regions so the whole of the world ends up being "land". Setting `clip=FALSE` disables this action, which may be of benefit in rare instances in the line connecting two points on a coastline may cross the plot domain, even if those points are outside that domain.

`type` indication of type; may be `"polygon"`, for a filled polygon, `"p"` for points, `"l"` for line segments, or `"o"` for points overlain with line segments.

`axes` logical value indicating whether to draw longitude and latitude values in the lower and left margin, respectively. This may not work well for some projections or scales.

`cex` character expansion factor for plot symbols, used if `type='p'` or any other value that yields symbols.

`cex.axis`	axis-label expansion factor (see `par`).
`mgp`	three-element numerical vector describing axis-label placement, passed to `mapAxis`.
`drawBox`	logical value indicating whether to draw a box around the plot. This is helpful for many projections at sub-global scale.
`showHemi`	logical value indicating whether to show the hemisphere in axis tick labels.
`polarCircle`	a number indicating the number of degrees of latitude extending from the poles, within which zones are not drawn.
`lonlabel, latlabel, sides`	Optional vectors of longitude and latitude to label on the indicated sides of plot, passed to `plot,coastline-method`. Using these arguments permits reasonably simple customization. If they are are not provided, reasonable defaults will be used.
`projection`	optional indication of projection, in one of two forms. First, it may be a character string in the "CRS" format that is used by the `rgdal` package (and in much of modern computer-based cartography). For example, `projection="+proj=merc"` specifies a Mercator projection. The second format is the output from `CRS` in the **sp** package, which is an object with a slot named `projarg` that gets used as a projection string. See "Details".
`tissot`	logical value indicating whether to use `mapTissot` to plot Tissot indicatrices, i.e. ellipses at grid intersection points, which indicate map distortion.
`trim`	logical value indicating whether to trim islands or lakes containing only points that are off-scale of the current plot box. This solves the problem of Antarctica overfilling the entire domain, for an Arctic-centred stereographic projection. It is not a perfect solution, though, because the line segment joining two off-scale points might intersect the plotting box.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	optional arguments passed to some plotting functions. This can be useful in many ways, e.g. Example 5 shows how to use `xlim` etc to reproduce a scale exactly between two plots.

Details

Creates a map using the indicated projection. As noted in the information on the `projection` argument, projections are specified in the notation used by `project()` in the `rgdal` package; see "Available Projections" for a list of possibilities.

Once a projection is set, other `map*` functions may be used to add to the map.

Further details on map projections are provided by [1,11], an exhaustive treatment that includes many illustrations, an overview of the history of the topic, and some notes on the strengths and weaknesses of the various formulations. See especially pages 2 through 7, which define terms and provide recommendations. Reference [2] is also useful, especially regarding datum shifts; [3] and [4] are less detailed and perhaps better for novices. See [8] for a gallery of projections.

Available Projections

Map projections are provided by the rgdal package, but not all projections in that package are available. The available list is given in the table below. The cartographic community has set up a naming scheme in a coded scheme, e.g. projection="+proj=aea" selects the Albers equal area projection.

The allowed projections include those PROJ.4 projections provided by rgdal that have inverses, minus a few that cause problems: alsk overdraws coastlineWorld, and is a niche projection for Alaska; calcofi is not a real projection, but rather a coordinate system; gs48 overdraws coastlineWorld, and is a niche projection for the USA; gs50 overdraws coastlineWorld, and is a niche projection for the USA; gstmerc overdraws coastlineWorld; isea causes segmentation faults on OSX systems; krovak overdraws coastlineWorld, and is a niche projection for the Czech Republic; labrd returns NaN for most of the world, and is a niche projection for Madagascar; lee_os overdraws coastlineWorld; and nzmg overdraws coastlineWorld.

The information in the table is reformatted from the output of the unix command proj -lP, where proj is provided by version 4.9.0 of the PROJ.4 system. Most of the arguments listed have default values. In addition, most projections can handle arguments lon_0 and lat_0, for shifting the reference point, although in some cases shifting the longitude can yield poor filling of coastlines.

Further details of the projections and the controlling arguments are provided at several websites, because PROJ.4 has been incorporated into rgdal and other R packages, plus many other software systems; a good starting point for learning is [6].

See “Examples” for suggested projections for some common applications, and [8] for a gallery indicating how to use every projection.

Projection	**Code**	**Arguments**
Albers equal area	aea	lat_1, lat_2
Azimuthal equidistant	aeqd	lat_0, guam
Aitoff	aitoff	-
Mod. stererographics of Alaska	alsk	-
Bipolar conic of western hemisphere	bipc	-
Bonne Werner	bonne	lat_1
Cassini	cass	-
Central cylindrical	cc	-
Equal area cylindrical	cea	lat_ts
Collignon	collg	-
Craster parabolic Putnins P4	crast	-
Eckert I	eck1	-
Eckert II	eck2	-
Eckert III	eck3	-
Eckert IV	eck4	-
Eckert V	eck5	-
Eckert VI	eck6	-
Equidistant cylindrical plate (Caree)	eqc	lat_ts, lat_0
Equidistant conic	eqdc	lat_1, lat_2
Euler	euler	lat_1, lat_2
Extended transverse Mercator	etmerc	lat_ts, lat_0
Fahey	fahey	-
Foucaut	fouc	-

Foucault sinusoidal	fouc_s	-
Gall stereographic	gall	-
Geostationary satellite view	geos	h
General sinusoidal series	gn_sinu	m, n
Gnomonic	gnom	-
Goode homolosine	goode	-
Hatano asymmetrical equal area	hatano	-
HEALPix	healpix	-
rHEALPix	rhealpix	north_square, south_square
Interrupted Goode homolosine	igh	-
Kavraisky V	kav5	-
Kavraisky VII	kav7	-
Lambert azimuthal equal area	laea	-
Longitude and latitude	lonlat	-
Longitude and latitude	longlat	-
Longitude and latitude	latlon	-
Lambert conformal conic	lcc	lat_1, lat_2, lat_0
Lambert equal area conic	leac	lat_1, south
Loximuthal	loxim	
Space oblique for Landsat	lsat	lsat, path
McBryde-Thomas flat-polar sine, no. 1	mbt_s	
McBryde-Thomas flat-polar sine, no. 2	mbt_fps	
McBryde-Thomas flat-polar parabolic	mbtfpp	
McBryde-Thomas flat-polar quartic	mbtfpq	
McBryde-Thomas flat-polar sinusoidal	mbtfps	
Mercator	merc	lat_ts
Miller oblated stereographic	mil_os	
Miller cylindrical	mill	
Mollweide	moll	
Murdoch I	murd1	lat_1, lat_2
Murdoch II	murd2	lat_1, lat_2
murdoch III	murd3	lat_1, lat_2
Natural earth	natearth	
Nell	nell	
Nell-Hammer	nell_h	
Near-sided perspective	nsper	h
New Zealand map grid	nzmg	
General oblique transformation	ob_tran	o_proj, o_lat_p, o_lon_p, o_alpha, o_lon_c o_lat_c, o_lon_1, o_lat_1, o_lon_2, o_lat_2
Oblique cylindrical equal area	ocea	lat_1, lat_2, lon_1, lon_2
Oblated equal area	oea	n, m, theta
Oblique Mercator	omerc	alpha, gamma, no_off, lonc, lon_1, lat_1, lon_2, lat_2
Orthographic	ortho	-
Perspective conic	pconic	lat_1, lat_2
Polyconic American	poly	-
Putnins P1	putp1	-
Putnins P2	putp2	-

Putnins P3	putp3	-
Putnins P3’	putp3p	-
Putnins P4’	putp4p	-
Putnins P5	putp5	-
Putnins P5’	putp5p	-
Putnins P6	putp6	-
Putnins P6’	putp6p	-
Quartic authalic	qua_aut	-
Quadrilateralized spherical cube	qsc	-
Robinson	robin	-
Roussilhe stereographic	rouss	-
Sinusoidal aka Sanson-Flamsteed	sinu	-
Swiss. oblique Mercator	somerc	-
Stereographic	stere	lat_ts
Oblique stereographic alternative	sterea	-
Transverse cylindrical equal area	tcea	-
Tissot	tissot	lat_1, lat_2
Transverse Mercator	tmerc	-
Two point equidistant	tpeqd	lat_1, lon_1, lat_2, lon_2
Tilted perspective	tpers	tilt, azi, h
Universal polar stereographic	ups	south
Urmaev flat-polar sinusoidal	urmfps	n
Universal transverse Mercator	utm	zone, south
van der Grinten I	vandg	-
Vitkovsky I	vitk1	lat_1, lat_2
Wagner I Kavraisky VI	wag1	-
Wagner II	wag2	-
Wagner III	wag3	lat_ts
Wagner IV	wag4	-
Wagner V	wag5	-
Wagner VI	wag6	-
Werenskiold I	weren	-
Winkel I	wink1	lat_ts
Winkel Tripel	wintri	lat_ts

Available ellipse formulations

In the PROJ.4 system of specifying projections, the following ellipse models are available: MERIT, SGS85, GRS80, IAU76, airy, APL4.9, NWL9D, mod_airy, andrae, aust_SA, GRS67, bessel, bess_nam, clrk66, clrk80, clrk80ign, CPM, delmbr, engelis, evrst30, evrst48, evrst56, evrst69, evrstSS, fschr60, fschr60m, fschr68, helmert, hough, intl, krass, kaula, lerch, mprts, new_intl, plessis, SEasia, walbeck, WGS60, WGS66, WGS72, WGS84, and sphere (the default). For example, use projection="+proj=aea +ellps=WGS84" for an Albers Equal Area projection using the most recent of the World Geodetic System model. It is unlikely that changing the ellipse will have a visible effect on plotted material at the plot scale appropriate to most oceanographic applications.

Available datum formulations

In the PROJ.4 system of specifying projections, the following datum formulations are available: `WGS84`, `GGRS87`, `Greek_Geodetic_Reference_System_1987`, `NAD83`, `North_American_Datum_1983`, `NAD27`, `North_American_Datum_1927`, `potsdam`, `Potsdam`, `carthage`, `Carthage`, `hermannskogel`, `Hermannskogel`, `ire65`, `Ireland`, `nzgd49`, `New`, `OSGB36`, and `Airy`. It is unlikely that changing the datum will have a visible effect on plotted material at the plot scale appropriate to most oceanographic applications.

Choosing a projection

The best choice of projection depends on the application. Readers may find `projection="+proj=moll"` useful for world-wide plots, `ortho` for hemispheres viewed from the equator, `stere` for polar views, `lcc` for wide meridional ranges in mid latitudes, and `merc` in limited-area cases where angle preservation is important.

Issues

Map projection is a complicated matter that is addressed here in a limited and pragmatic way. For example, `mapPlot` tries to draw axes along a box containing the map, instead of trying to find spots along the "edge" of the map at which to put longitude and latitude labels. This design choice greatly simplifies the coding effort, freeing up time to work on issues regarded as more pressing. Chief among those issues are (a) the occurrence of horizontal lines in maps that have prime meridians (b) inaccurate filling of land regions that (again) occur with shifted meridians and (c) inaccurate filling of Antarctica in some projections. Generally, issues are tackled first for commonly used projections, such as those used in the examples.

Changes

- 2017-11-19: `imw_p` removed, because it has problems doing inverse calculations. This is a also problem in the standalone PROJ.4 application version 4.9.3, downloaded and built on OSX. See https://github.com/dankelley/oce/issues/1319 for details.
- 2017-11-17: `lsat` removed, because it does not work in `rgdal` or in the latest standalone PROJ.4 application. This is a also problem in the standalone PROJ.4 application version 4.9.3, downloaded and built on OSX. See https://github.com/dankelley/oce/issues/1337 for details.
- 2017-09-30: `lcca` removed, because its inverse was wildly inaccurate in a Pacific Antarctic-Alaska application (see https://github.com/dankelley/oce/issues/1303).

Author(s)

Dan Kelley and Clark Richards

References

1. Snyder, John P., 1987. Map Projections: A Working Manual. USGS Professional Paper: 1395 (available at pubs.usgs.gov/pp/1395/report.pdf).

2. Natural Resources Canada http://www.nrcan.gc.ca/earth-sciences/geography/topographic-information/maps/9805

3. Wikipedia page http://en.wikipedia.org/wiki/List_of_map_projections

4. Radical Cartography website http://www.radicalcartography.net/?projectionref (This URL worked prior to Nov 16, 2016, but was found to fail on that date.)

5. The PROJ.4 website is http://trac.osgeo.org/proj, and it is the place to start to learn about the code.

6. PROJ.4 projection details were once at http://www.remotesensing.org/geotiff/proj_list/ but it was discovered on Dec 18, 2016, that this link no longer exists. Indeed, there seems to have been significant reorganization of websites related to this. The base website seems to be https://trac.osgeo.org/geotiff/ and that lists only what is called an unofficial listing, on the wayback web-archiver server http://web.archive.org/web/20160802172057/http://www.remotesensing.org/geotiff/proj_list/

7. A gallery of map plots is provided at http://dankelley.github.io/r/2015/04/03/oce-proj.html.

8. A fascinating historical perspective is provided by Snyder, J. P. (1993). Two thousand years of map projections. University of Chicago Press.

See Also

Points may be added to a map with mapPoints, lines with mapLines, text with mapText, polygons with mapPolygon, images with mapImage, and scale bars with mapScalebar. Points on a map may be determined with mouse clicks using mapLocator. Great circle paths can be calculated with geodGc. See [8] for a demonstration of the available map projections (with graphs).

Other functions related to maps: lonlat2map, lonlat2utm, map2lonlat, mapArrows, mapAxis, mapContour, mapDirectionField, mapGrid, mapImage, mapLines, mapLocator, mapLongitudeLatitudeXY, mapPoints, mapPolygon, mapScalebar, mapText, mapTissot, oceCRS, shiftLongitude, usrLonLat, utm2lonlat

Examples

```
## Not run:
library(oce)
data(coastlineWorld)

# Example 1.
# Mollweide ([1] page 54) is an equal-area projection that works well
# for whole-globe views, below shown in a Pacific-focus view.
# Note that filling is not employed when the prime meridian
# is shifted, because this causes a problem with Antarctica
par(mfrow=c(2, 1), mar=c(3, 3, 1, 1))
mapPlot(coastlineWorld, projection="+proj=moll", col='gray')
mtext("Mollweide", adj=1)
cl180 <- coastlineCut(coastlineWorld, lon_0=-180)
mapPlot(cl180, projection="+proj=moll +lon_0=-180")
mtext("Mollweide", adj=1)
par(mfrow=c(1, 1))

# Example 2.
# Orthographic projections resemble a globe, making them attractive for
# non-technical use, but they are neither conformal nor equal-area, so they
```

```
# are somewhat limited for serious use on large scales.  See Section 20 of
# [1]. Note that filling is not employed because it causes a problem with
# Antarctica.
par(mar=c(3, 3, 1, 1))
mapPlot(coastlineWorld, projection="+proj=ortho +lon_0=-180")
mtext("Orthographic", adj=1)

# Example 3.
# The Lambert conformal conic projection is an equal-area projection
# recommended by [1], page 95, for regions of large east-west extent
# away from the equator, here illustrated for the USA and Canada.
par(mar=c(3, 3, 1, 1))
mapPlot(coastlineCut(coastlineWorld, -100),
        longitudelim=c(-130,-55), latitudelim=c(35, 60),
        projection="+proj=lcc +lat_0=30 +lat_1=60 +lon_0=-100", col='gray')
mtext("Lambert conformal", adj=1)

# Example 4.
# The stereographic projection [1], page 120, is conformal, used
# below for an Arctic view with a Canadian focus.  Note the trick of going
# past the pole: the second latitudelim value is 180 minus the first, and the
# second longitudelim is 180 plus the first; this uses image points "over"
# the pole.
par(mar=c(3, 3, 1, 1))
mapPlot(coastlineCut(coastlineWorld, -135),
        longitudelim=c(-130, 50), latitudelim=c(70, 110),
        proj="+proj=stere +lat_0=90 +lon_0=-135", col='gray')
mtext("Stereographic", adj=1)

# Example 5.
# Spinning globe: create PNG files that can be assembled into a movie
png("globe-%03d.png")
lons <- seq(360, 0, -15)
par(mar=rep(0, 4))
for (i in seq_along(lons)) {
    p <- paste("+proj=ortho +lat_0=30 +lon_0=", lons[i], sep="")
    if (i == 1) {
        mapPlot(coastlineCut(coastlineWorld, lons[i]),
                projection=p, col="lightgray")
        xlim <- par("usr")[1:2]
        ylim <- par("usr")[3:4]
    } else {
        mapPlot(coastlineCut(coastlineWorld, lons[i]),
                projection=p, col="lightgray",
                xlim=xlim, ylim=ylim, xaxs="i", yaxs="i")
    }
}
dev.off()

## End(Not run)
```

mapPoints *Add Points to a Map*

Description

Plot points on an existing map.

Usage

```
mapPoints(longitude, latitude, debug = getOption("oceDebug"), ...)
```

Arguments

`longitude`	Longitudes of points to be plotted, or an object from which longitude and latitude can be inferred in which case the following two arguments are ignored. This objects that are possible include those of type `coastline`.
`latitude`	Latitudes of points to be plotted.
`debug`	A flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	Optional arguments passed to `points`.

Details

Adds points to an existing map, by analogy to `points`.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, longitudelim=c(-80, 0), latitudelim=c(20, 50),
        col="lightgray", projection="+proj=laea +lon_0=-35")
data(section)
mapPoints(section)
```

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

---

mapPolygon *Add a Polygon to a Map*

Description

Plot a polygon on an existing map.

Usage

```
mapPolygon(longitude, latitude, density = NULL, angle = 45, border = NULL,
  col = NA, lty = par("lty"), ..., fillOddEven = FALSE)
```

Arguments

`longitude`	longitudes of points to be plotted, or an object from which longitude and latitude can be inferred (e.g. a coastline file, or the return value from `mapLocator`), in which case the following two arguments are ignored.
`latitude`	latitudes of points to be plotted.
`density`	as for `polygon`.
`angle`	as for `polygon`.
`border`	as for `polygon`.
`col`	as for `polygon`.
`lty`	as for `polygon`.
`...`	as for `polygon`.
`fillOddEven`	as for `polygon`.

Details

Adds a polygon to an existing map, by analogy to `polygon`. Used by `mapImage`.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

mapScalebar — *Add a Scalebar to a Map*

Description

Draw a scalebar on an existing map.

Usage

```
mapScalebar(x, y = NULL, length, lwd = 1.5 * par("lwd"), cex = par("cex"),
  col = "black")
```

Arguments

`x, y`	position of the scalebar. Eventually this may be similar to the corresponding arguments in `legend`, but at the moment y must be `NULL` and x must be `"topleft"`.
`length`	the distance to indicate, in kilometres. If not provided, a reasonable choice is made, based on the underlying map.
`lwd`	line width of the scalebar.
`cex`	character expansion factor for the scalebar text.
`col`	colour of the scalebar.

Details

The scale is appropriate to the centre of the plot, and will become increasingly inaccurate away from that spot, with the error depending on the projection and the fraction of the earth that is shown.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
## Arctic Ocean
par(mar=c(2.5, 2.5, 1, 1))
mapPlot(coastlineWorld, latitudelim=c(60, 120), longitudelim=c(-130,-50),
```

```
        col="lightgray", projection="+proj=stere +lat_0=90")
mapScalebar()

## End(Not run)
```

---

mapText *Add Text to a Map*

Description

Plot text on an existing map.

Usage

```
mapText(longitude, latitude, labels, ...)
```

Arguments

`longitude`	vector of longitudes of text to be plotted.
`latitude`	vector of latitudes of text to be plotted.
`labels`	vector of labels of text to be plotted.
`...`	optional arguments passed to `text`, e.g. adj, pos, etc.

Details

Adds text to an existing map, by analogy to `text`.

Author(s)

Dan Kelley

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
longitude <- coastlineWorld[['longitude']]
latitude <- coastlineWorld[['latitude']]
mapPlot(longitude, latitude, type='l', grid=5,
        longitudelim=c(-70,-50), latitudelim=c(45, 50),
```

```
        projection="+proj=merc")
lon <- -63.5744 # Halifax
lat <- 44.6479
mapPoints(lon, lat, pch=20, col="red")
mapText(lon, lat, "Halifax", col="red", pos=1, offset=1)

## End(Not run)
```

`mapTissot` *Add Tissot Indicatrices to a Map*

Description

Plot ellipses at grid intersection points, as a method for indicating the distortion inherent in the projection [1]. (Each ellipse is drawn with 64 segments.)

Usage

```
mapTissot(grid = rep(15, 2), scale = 0.2, crosshairs = FALSE, ...)
```

Arguments

`grid`	numeric vector of length 2, specifying the increment in longitude and latitude for the grid. Indicatrices are drawn at e.g. longitudes `seq(-180, 180, grid[1])`.
`scale`	numerical scale factor for ellipses. This is multiplied by `min(grid)` and the result is the radius of the circle on the earth, in latitude degrees.
`crosshairs`	logical value indicating whether to draw constant-latitude and constant-longitude crosshairs within the ellipses. (These are drawn with 10 line segments each.) This can be helpful in cases where it is not desired to use `mapGrid` to draw the longitude/latitude grid.
`...`	extra arguments passed to plotting functions, e.g. `col="red"` yields red indicatrices.

Details

The purpose and interpretation are outlined in [1], but should also be self-explanatory.

Author(s)

Dan Kelley

References

1. Snyder, John P., 1987. Map Projections: A Working Manual. USGS Professional Paper: 1395 (available at `pubs.usgs.gov/pp/1395/report.pdf`).

See Also

A map must first have been created with `mapPlot`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `oceCRS`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
par(mfrow=c(1, 1), mar=c(2, 2, 1, 1))
p  <- "+proj=aea +lat_1=10 +lat_2=60 +lon_0=-45"
mapPlot(coastlineWorld, projection=p, col="gray",
longitudelim=c(-90,0), latitudelim=c(0, 50))
mapTissot(c(15, 15), col='red')

## End(Not run)
```

`mapZones` *Add Zones to a Map [deprecated]*

Description

WARNING: This function will be removed soon; see oce-deprecated.

Usage

```
mapZones(longitude, polarCircle = 0, lty = "solid", lwd = 0.5 *
  par("lwd"), col = "darkgray", ...)
```

Arguments

`longitude`	either a logical indicating whether to draw a zonal grid, or a vector of longitudes at which to draw zones.
`polarCircle`	a number indicating the number of degrees of latitude extending from the poles, within which zones are not drawn.
`lty`	line type.
`lwd`	line width.
`col`	line colour.
`...`	optional arguments passed to `lines`.

Details

Use mapGrid instead of the present function.

Plot zones (lines of constant longitude) on a existing map.

This function should not be used, since it will be removed soon. Please use mapGrid() instead.

Author(s)

Dan Kelley

See Also

Other functions that will be removed soon: addColumn, ctdAddColumn, ctdUpdateHeader, mapMeridians, oce.as.POSIXlt

---

matchBytes *Locate byte sequences in a raw vector*

Description

Find spots in a raw vector that match a given byte sequence.

Usage

```
matchBytes(input, b1, ...)
```

Arguments

input	a vector of raw (byte) values.
b1	a vector of bytes to match (must be of length 2 or 3 at present; for 1-byte, use which).
...	additional bytes to match for (up to 2 permitted)

Value

List of the indices of `input` that match the start of the `bytes` sequence (see example).

Author(s)

Dan Kelley

Examples

```
buf <- as.raw(c(0xa5, 0x11, 0xaa, 0xa5, 0x11, 0x00))
match <- matchBytes(buf, 0xa5, 0x11)
print(buf)
print(match)
```

matrixShiftLongitude *Rearrange areal matrix so Greenwich is near the centre*

Description

Sometimes datasets are provided in matrix form, with first index corresponding to longitudes ranging from 0 to 360. `matrixShiftLongitude` cuts such matrices at longitude=180, and swaps the pieces so that the dateline is at the left of the matrix, not in the middle.

Usage

```
matrixShiftLongitude(m, longitude)
```

Arguments

`m`	The matrix to be modified.
`longitude`	A vector containing the longitude in the 0-360 convention. If missing, this is constructed to range from 0 to 360, with as many elements as the first index of `m`.

Value

A list containing `m` and `longitude`, both rearranged as appropriate.

See Also

`shiftLongitude` and `standardizeLongitude`.

matrixSmooth *Smooth a Matrix*

Description

The values on the edge of the matrix are unaltered. For interior points, the result is defined in terms in terms of the original as follows. $r_{i,j} = (2m_{i,j} + m_{i-1,j} + m_{i+1,j} + m_{i,j-1} + m_{i,j+1})/6$. Note that missing values propagate to neighbours.

Usage

```
matrixSmooth(m, passes = 1)
```

Arguments

`m`	a matrix to be smoothed.
`passes`	an integer specifying the number of times the smoothing is to be applied.

Value

A smoothed matrix.

Author(s)

Dan Kelley

Examples

```
library(oce)
opar <- par(no.readonly = TRUE)
m <- matrix(rep(seq(0, 1, length.out=5), 5), nrow=5, byrow=TRUE)
m[3, 3] <- 2
m1 <- matrixSmooth(m)
m2 <- matrixSmooth(m1)
m3 <- matrixSmooth(m2)
par(mfrow=c(2, 2))
image(m,  col=rainbow(100), zlim=c(0, 4), main="original image")
image(m1, col=rainbow(100), zlim=c(0, 4), main="smoothed 1 time")
image(m2, col=rainbow(100), zlim=c(0, 4), main="smoothed 2 times")
image(m3, col=rainbow(100), zlim=c(0, 4), main="smoothed 3 times")
par(opar)
```

`met` *Sample met Object*

Description

This is sample `met` object containing data for Halifax, Nova Scotia, during September of 2003 (the period during which Hurricane Juan struck the city).

Details

The data file was downloaded with

```
metFile <- download.met(id=6358, year=2003, month=9, destdir=".")
met <- read.met(metFile)
met <- oceSetData(met, "time", met[["time"]]+4*3600,
                  note="add 4h to local time to get UTC time")
```

Using `download.met` avoids having to navigate the the awkward Environment Canada website, but it imposes the burden of having to know the station number. See the documentation for `download.met` for more details on station numbers.

Source

Environment Canada website on February 1, 2017

See Also

Other datasets provided with oce: adp, adv, argo, cm, coastlineWorld, colors, ctdRaw, ctd, echosounder, landsat, lisst, lobo, rsk, sealevelTuktoyaktuk, sealevel, section, topoWorld, wind

Other things related to met data: [[,met-method, [[<-,met-method, as.met, download.met, met-class, plot,met-method, read.met, subset,met-method, summary,met-method

---

met-class *Class to Store Meteorological Data*

Description

Class to store meteorological data, with standard slots metadata, data and processingLog. For objects created with read.met, the data slot will contain all the columns within the original file (with some guesses as to units) in addition to several calculated quantities such as u and v, which are velocities in m/s (not the km/h stored in typical data files), and which obey the oceanographic convention that u>0 is a wind towards the east.

Methods

Accessing values. For an object named m, temperature (in degC) may be accessed as m[["temperature"]], dew point (in degC) as m[["dewPoint"]], pressure (in kPa) as m[["pressure"]], eastward wind component (in m/s) as m[["u"]], northward wind component (in m/s) as m[["v"]], and wind direction (in degrees clockwise of North) as "direction". The filename from which the data came (if any) may be found with m[["filename"]]. Items in metadata must be specifield by full name, but those in data may be abbreviated, so long as the abbreviation is unique.

Assigning values. Everything that may be accessed may also be assigned, e.g.

```
m[["temperature"]] <- 1 + m[["temperature"]]
```

increases temperature by 1C.

Author(s)

Dan Kelley

See Also

Other classes provided by oce: adp-class, adv-class, argo-class, bremen-class, cm-class, coastline-class, ctd-class, echosounder-class, lisst-class, lobo-class, oce-class, odf-class, rsk-class, sealevel-class, section-class, topo-class, windrose-class

Other things related to met data: [[,met-method, [[<-,met-method, as.met, download.met, met, plot,met-method, read.met, subset,met-method, summary,met-method

metNames2oceNames	*Convert met Data Name to Oce Name*

Description

Convert met Data Name to Oce Name

Usage

```
metNames2oceNames(names, scheme)
```

Arguments

`names`	a vector of character strings with original names
`scheme`	an optional indication of the scheme that is employed. This may be `"ODF"`, in which case `ODFNames2oceNames` is used, or `"met"`, in which case some tentative code for met files is used.

Details

Interoperability between oce functions requires that standardized data names be used, e.g. `"temperature"` for in-situ temperature. Very few data-file headers name the temperature column in exactly that way, however, and this function is provided to try to guess the names. The task is complicated by the fact that Environment Canada seems to change the names of the columns, e.g. sometimes a symbol is used for the degree sign, other times not.

Several quantities in the returned object differ from their values in the source file. For example, speed is converted from km/h to m/s, and angles are converted from tens of degres to degrees. Also, some items are created from scratch, e.g. `u` and `v`, the eastward and northward velocity, are computed from speed and direction. (Note that e.g. `u` is positive if the wind blows to the east; the data are thus in the normal Physics convention.)

Value

Vector of strings for the decoded names. If an unknown scheme is provided, this will just be `names`.

moonAngle	*Lunar Angle as Function of Space and Time*

Description

The calculations are based on formulae provided by Meeus [1982], primarily in chapters 6, 18, and 30. The first step is to compute sidereal time as formulated in Meeus [1982] chapter 7, which in turn uses Julian day computed according to as formulae in Meeus [1982] chapter 3. Using these quantities, formulae in Meeus [1982] chapter 30 are then used to compute geocentric longitude ($lambda$, in the Meeus notation), geocentric latitude ($beta$), and parallax. Then the `obliquity` of the ecliptic is computed with Meeus [1982] equation 18.4. Equatorial coordinates (right ascension and declination) are computed with equations 8.3 and 8.4 from Meeus [1982], using `eclipticalToEquatorial`. The hour angle (H) is computed using the unnumbered equation preceding Meeus's [1982] equation 8.1. Finally, Meeus [1982] equations 8.5 and 8.6 are used to calculate the local azimuth and altitude of the moon, using `equatorialToLocalHorizontal`.

Usage

```
moonAngle(t, longitude = 0, latitude = 0, useRefraction = TRUE)
```

Arguments

`t`	time, a POSIXt object (converted to timezone "UTC", if it is not already in that timezone), or a numeric value that corresponds to such a time.
`longitude`	observer longitude in degrees east
`latitude`	observer latitude in degrees north
`useRefraction`	boolean, set to `TRUE` to apply a correction for atmospheric refraction. (Ignored at present.)

Value

A list containing the following.

`time`	time
`azimuth`	moon azimuth, in degrees eastward of north, from 0 to 360. Note: this is not the convention used by Meeus, who uses degrees westward of South. (See diagram below.)
`altitude`	moon altitude, in degrees from -90 to 90. (See diagram below.)
`rightAscension`	right ascension, in degrees
`declination`	declination, in degrees
`lambda`	geocentric longitude, in degrees
`beta`	geocentric latitude, in degrees
`diameter`	lunar diameter, in degrees.
`distance`	earth-moon distance, in kilometers)
`illuminatedFraction`	fraction of moon's visible disk that is illuminated
`phase`	phase of the moon, defined in equation 32.3 of Meeus [1982]. The fractional part of which is 0 for new moon, 1/4 for first quarter, 1/2 for full moon, and 3/4 for last quarter.

Alternate formulations

Formulae provide by Meeus [1982] are used for all calculations here. Meeus [1991] provides formulae that are similar, but that differ in the 5th or 6th digits. For example, the formula for ephemeris time in Meeus [1991] differs from that in Meeus [1992] at the 5th digit, and almost all of the approximately 200 coefficients in the relevant formulae also differ in the 5th and 6th digits. Discussion of the changing formulations is best left to members of the astronomical community. For the present purpose, it may be sufficient to note that `moonAngle`, based on Meeus [1982], reproduces the values provided in example 45.a of Meeus [1991] to 4 significant digits, e.g. with all angles matching to under 2 minutes of arc.

Author(s)

Dan Kelley, based on formulae in Meeus [1982].

References

Meeus, Jean, 1982. Astronomical formulae for calculators. Willmann-Bell. Richmond VA, USA. 201 pages.

Meeus, Jean, 1991. Astronomical algorithms. Willmann-Bell, Richmond VA, USA. 429 pages.

See Also

The equivalent function for the sun is `sunAngle`.

Other things related to astronomy: `eclipticalToEquatorial`, `equatorialToLocalHorizontal`, `julianCenturyAnomaly`, `julianDay`, `siderealTime`, `sunAngle`

Examples

```
library(oce)
par(mfrow=c(3,2))
y <- 2012
m <- 4
days <- 1:3
## Halifax sunrise/sunset (see e.g. http://www.timeanddate.com/worldclock)
rises <- ISOdatetime(y, m, days,c(13,15,16), c(55, 04, 16),0,tz="UTC") + 3 * 3600 # ADT
sets <- ISOdatetime(y, m,days,c(3,4,4), c(42, 15, 45),0,tz="UTC") + 3 * 3600
azrises <- c(69, 75, 82)
azsets <- c(293, 288, 281)
latitude <- 44.65
longitude <- -63.6
for (i in 1:3) {
    t <- ISOdatetime(y, m, days[i],0,0,0,tz="UTC") + seq(0, 24*3600, 3600/4)
    ma <- moonAngle(t, longitude, latitude)
    oce.plot.ts(t, ma$altitude, type='l', mar=c(2, 3, 1, 1), cex=1/2, ylab="Altitude")
    abline(h=0)
    points(rises[i], 0, col='red', pch=3, lwd=2, cex=1.5)
    points(sets[i], 0, col='blue', pch=3, lwd=2, cex=1.5)
    oce.plot.ts(t, ma$azimuth, type='l', mar=c(2, 3, 1, 1), cex=1/2, ylab="Azimuth")
    points(rises[i], -180+azrises[i], col='red', pch=3, lwd=2, cex=1.5)
```

```
    points(sets[i], -180+azsets[i], col='blue', pch=3, lwd=2, cex=1.5)
}
```

`numberAsHMS` *Convert a Numeric Time to Hour, Minute, and Second*

Description

Convert a Numeric Time to Hour, Minute, and Second

Usage

```
numberAsHMS(t, default = 0)
```

Arguments

`t`	a vector of factors or character strings, in the format 1200 for 12:00, 0900 for 09:00, etc.
`default`	value to be used for the returned hour, minute and second if there is something wrong with the input value (e.g. its length exceeds 4 characters, or it contains non-numeric characters)

Value

A list containing `hour`, `minute`, and `second`, the last of which is always zero.

Author(s)

Dan Kelley

See Also

Other things related to time: `ctimeToSeconds`, `julianCenturyAnomaly`, `julianDay`, `numberAsPOSIXct`, `secondsToCtime`, `unabbreviateYear`

Examples

```
t <- c("0900", "1234")
numberAsHMS(t)
```

numberAsPOSIXct *Convert a Numeric Time to a POSIXct Time*

Description

There are many varieties, according to the value of `type` as defined in 'Details'.

Usage

```
numberAsPOSIXct(t, type = c("unix", "matlab", "gps", "argo", "ncep1", "ncep2",
  "sas", "spss", "yearday"), tz = "UTC")
```

Arguments

`t`	an integer corresponding to a time, in a way that depends on `type`.
`type`	the type of time (see "Details").
`tz`	a string indicating the time zone, used only for unix and matlab times, since GPS times are always referenced to the UTC timezone.

Details

- `"unix"` employs Unix times, measured in seconds since the start of the year 1970.
- `"matlab"` employs Matlab times, measured in days since what MathWorks [1] calls "January 0, 0000" (i.e. `ISOdatetime(0, 1, 1, 0,0, 0)` in R notation).
- `"gps"` employs the GPS convention. For this, `t` is a two-column matrix, with the first column being the the GPS "week" (referenced to 1999-08-22) and the second being the GPS "second" (i.e. the second within the week). Since the GPS satellites do not handle leap seconds, the R-defined `.leap.seconds` is used for corrections.
- `"argo"` employs Argo times, measured in days since the start of the year 1900.
- `"ncep1"` employs NCEP times, measured in hours since the start of the year 1800.
- `"ncep2"` employs NCEP times, measured in days since the start of the year 1. (Note that, for reasons that are unknown at this time, a simple R expression of this definition is out by two days compared with the UDUNITS library, which is used by NCEP. Therefore, a two-day offset is applied. See [2, 3].)
- `"sas"` employs SAS times, indicated by `type="sas"`, have origin at the start of 1960.
- `"spss"` employs SPSS times, in seconds after 1582-10-14.
- `"yearday"` employs a convention in which `t` is a two-column matrix, with the first column being the year, and the second the yearday (starting at 1 for the first second of January 1, to match the convention used by Sea-Bird CTD software).

Value

A `POSIXct` time vector.

Author(s)

Dan Kelley

References

[1] Matlab times: http://www.mathworks.com/help/matlab/ref/datenum.html

[2] NCEP times: https://www.esrl.noaa.gov/psd/data/gridded/faq.html#3

[3] problem with NCEP times: https://github.com/dankelley/oce/issues/738

See Also

numberAsHMS

Other things related to time: ctimeToSeconds, julianCenturyAnomaly, julianDay, numberAsHMS, secondsToCtime, unabbreviateYear

Examples

```
numberAsPOSIXct(0)                      # unix time 0
numberAsPOSIXct(1, type="matlab")       # matlab time 1
numberAsPOSIXct(cbind(566, 345615), type="gps") # Canada Day, zero hour UTC
numberAsPOSIXct(cbind(2013, 0), type="yearday") # start of 2013
```

---

oce — *oce: A Package for Oceanographic Analysis*

Description

The oce package provides functions for working with Oceanographic data, for calculations that are specific to Oceanography, and for producing graphics that match the conventions of the field.

Specialized functions

A key function is read.oce, which will attempt to read Oceanographic data in raw format. This uses oceMagic to try to detect the file type, based on the file name and contents. If it proves impossible to detect the type, users should next try a more specialized function, e.g. read.ctd for CTD files, or read.ctd.sbe for Teledyne-Seabird files.

Generic methods

A list of the generic methods in oce is provided by ‘methods(class="oce")‘; a few that are used frequently are as follows.

[[Find the value of an item in the object's `metadata` or `data` slot. If the item does not exist, but can be calculated from the other items, then the calculated value is returned. As an example of the latter, consider the built-in `ctd` dataset, which does not contain potential temperature, `"theta"`. Using `ctd[["theta"]]` therefore causes `swTheta` to be called, to calculate `theta`. See [[,oce-method or type `?"[[,oce-method"` to learn more.

[[<- Alters the named item in the object's `metadata` or `data` slot. If the item does not exist, it is created. See [[<-,oce-method or type `?"[[<-,oce-method"` to learn more.

summary Displays some information about the object named as an argument, including a few elements from its `metadata` slot and some statistics of the contents of its `data` slot. See summary,oce-method or type `?"summary,oce-method"` to learn more.

subset Takes a subset of an oce object. See subset,oce-method or type `?"subset,oce-method"` to learn more.

Oceanographic data types handled

Over a dozen specialized data types are handled by oce, with generic plots and summaries for each, along with the specialized functions needed for typical Oceanographic analysis.

Oce object structure

See `oce-class` for a summary of the class structure and links to documentation for the many sub-classes of oce objects, each aligned with a class of instrument or or type of dataset.

`oce-class` *Base Class for oce Objects*

Description

This is mainly used within oce to create sub-classes, although users can use `new("oce")` to create a blank `oce` object, if desired.

Slots

`metadata` A list containing information about the data. The contents vary across sub-classes, e.g. an `adp-class` object has information about beam patterns, which obviously would not make sense for a `ctd-class` object. In addition, all classes have items named `units` and `flags`, used to store information on the units of the data, and the data quality.

`data` A list containing the data.

`processingLog` A list containing time-stamped processing steps, typically stored in the object by oce functions.

See Also

Other classes provided by `oce`: `adp-class`, `adv-class`, `argo-class`, `bremen-class`, `cm-class`, `coastline-class`, `ctd-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `odf-class`, `rsk-class`, `sealevel-class`, `section-class`, `topo-class`, `windrose-class`

Examples

```
str(new("oce"))
```

`oce-deprecated` *Deprecated and Defunct Elements of package 'oce'*

Description

Certain functions and function arguments are still provided for compatibility with older versions of 'oce', but will be removed soon. The 'oce' scheme for removing functions is similar to that used by 'Bioconductor': items are marked as "deprecated" in one release, marked as "defunct" in the next, and removed in the next after that. This goal is to provide a gentle migration path for users who keep their packages reasonably up-to-date.

Details

Several 'oce' functions are marked "deprecated" in the present release of oce. Please use the replacement functions as listed below. The next CRAN release of 'oce' will designate these functions as "defunct".

Deprecated	**Replacement**	**Notes**
`findInOrdered(x,f)`	`findInterval(f,x)`	Deprecated 2017-09-07
`mapZones`	`mapGrid`	Deprecated 2016-02-13
`mapMeridians`	`mapGrid`	Deprecated 2016-02-13
`addColumn`	`oceSetData`	Deprecated 2016-08-01
`oce.magic`	`oceMagic`	Deprecated 2016-09-01
`ctdAddColumn`	`oceSetData`	Deprecated 2016-11-11
`ctdUpdateHeader`	-	Deprecated 2016-11-11
`oce.as.POSIXlt`	`parse_date_time`	Deprecated 2016-12-17

The following are marked "defunct", so calling them in the the present version produces an error message that hints at a replacement function. Once a function is marked "defunct" on one CRAN release, it will be slated for outright deletion in a subsequent release.

Defunct	**Replacement**	**Notes**
`makeSection`	`as.section`	Improve utility and name sensibility
`columns`	`read.ctd`	Unnecessary; never worked

The following were removed in recent years.

Function	**Replacement**	**Notes**
FILL IN	FILL IN	FILL IN

Several 'oce' function arguments are considered "deprecated", which means they will be marked "defunct" in the next CRAN release. These are normally listed in the help page for the function in question. A few that may be of general interest are also listed below.

- The `endian` argument of `byteToBinary` will be removed sometime in the year 2017, and should be set to `"big"` in the meantime.
- The `parameters` argument of `plot,ctd-method` was deprecated on 2016-12-30. It was once used by `plot,coastline-method` but has been ignored by that function since February 2016.
- The `orientation` argument of `plot,ctd-method` was deprecated on 2016-12-30. It was once used by `plot,coastline-method` but has been ignored by that function since February 2016.

Several 'oce' function arguments are considered "defunct", which means they will be removed in the next CRAN release. They are as follows.

- The `date` argument of `as.ctd` was discovered to have been unused in early 2016. Since the `startTime` actually fills its role, `date` was considered to be deprecated in June 2016.
- The `quality` flag of `as.ctd` was marked as deprecated in March 2016.
- The `fill` argument of `mapPlot` was confusing to users, so it was designated as deprecated in June 2016. (The confusion stemmed from subtle differences between `plot` and `polygon`, and the problem is that `mapPlot` can use either of these functions, according to whether coastlines are to be filled.) The functionality is preserved, in the `col` argument.

See Also

The 'Bioconductor' scheme for removing functions is described at `https://www.bioconductor.org/developers/how-to/deprecation/` and it is extended here to function arguments.

`oce.as.POSIXlt` *Oce Variant of as.POSIXlt [deprecated]*

Description

WARNING: This function will be removed soon; see oce-deprecated. It was realized in December of 2016 that this function was not used within oce, and also that `parse_date_time` in the **lubridate** package was superior and therefore a better choice for "oce" users.

Usage

```
oce.as.POSIXlt(x, tz = "")
```

Arguments

`x`	a date, as for `as.POSIXlt`, but also including forms in which the month name appears.
`tz`	the timezone, as for `as.POSIXlt`

Value

A POSIXlt object.

Author(s)

Dan Kelley

See Also

Other functions that will be removed soon: `addColumn`, `ctdAddColumn`, `ctdUpdateHeader`, `mapMeridians`, `mapZones`

`oce.as.raw` *Version of as.raw() that clips data*

Description

A version of as.raw() that clips data to prevent warnings

Usage

```
oce.as.raw(x)
```

Arguments

`x`	values to be converted to raw

Details

Negative values are clipped to 0, while values above 255 are clipped to 255; the result is passed to `as.raw` and returned.

Value

Raw values corresponding to `x`.

Author(s)

Dan Kelley

Examples

```
x <- c(-0.1, 0, 1, 255, 255.1)
data.frame(x, oce.as.raw(x))
```

`oce.axis.POSIXct` *Oce Version of axis.POSIXct*

Description

A specialized variant of `axis.POSIXct` that produces results with less ambiguity in axis labels.

Usage

```
oce.axis.POSIXct(side, x, at, tformat, labels = TRUE, drawTimeRange,
  abbreviateTimeRange = FALSE, drawFrequency = FALSE, cex = par("cex"),
  cex.axis = par("cex.axis"), cex.main = par("cex.main"),
  mar = par("mar"), mgp = par("mgp"), main = "",
  debug = getOption("oceDebug"), ...)
```

Arguments

`side`	as for `axis.POSIXct`.
`x`	as for `axis.POSIXct`.
`at`	as for `axis.POSIXct`.
`tformat`	as `format` for `axis.POSIXct` for now, but may eventually have new features for multiline labels, e.g. day on one line and month on another.
`labels`	as for `axis.POSIXct`.
`drawTimeRange`	Optional indication of whether/how to draw the time range in the margin on the side of the the plot opposite the time axis. If this is not supplied, it defaults to the value returned by `getOption("oceDrawTimeRange")`, and if that option is not set, it defaults to `TRUE`. No time range is drawn if `drawTimeRange` is `FALSE`. If it is `TRUE`, the range will be shown. This range refers to range of the x axis (not the data). The format of the elements of that range is set by `getOption("oceTimeFormat")` (or with the default value of an empty string, if this option has not been set). The timezone will be indicated if the time range is under a week. For preliminary work, it makes sense to use `drawTimeRange=TRUE`, but for published work it can be better to drop this label and indicate something about the time in the figure caption.
`abbreviateTimeRange`	boolean, `TRUE` to abbreviate the second number in the time range, e.g. dropping the year if it is the same in the first number.
`drawFrequency`	boolean, `TRUE` to show the frequency of sampling in the data
`cex`	size of labels on axes; see `par`("cex").
`cex.axis`	see `par`("cex.axis").
`cex.main`	see `par`("cex.main").
`mar`	value for `par(mar)` for axis
`mgp`	value for `par(mgp)` for axis

`main`	title of plot
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	as for `axis.POSIXct`.

Details

The tick marks are set automatically based on examination of the time range on the axis. The scheme was devised by constructing test cases with a typical plot size and font size, and over a wide range of time scales. In some categories, both small tick marks are interspersed between large ones.

The user may set the format of axis numbers with the `tformat` argument. If this is not supplied, the format is set based on the time span of the axis:

- If this time span is less than a minute, the time axis labels are in seconds (fractional seconds, if the interval is less than 2 seconds), with leading zeros on small integers. (Fractional seconds are enabled with a trick: the usual R format `"%S"` is supplemented with a new format e.g. `"%.2S"`, meaning to use two digits after the decimal.)
- If the time span exceeds a minute but is less than 1.5 days, the label format is `"%H:%M:%S"`.
- If the time span exceeds 1.5 days but is less than 1 year, the format is `"%b %d"` (e.g. Jul 15) and, again, the tick marks are set up for several subcategories.
- If the time span exceeds a year, the format is `"%Y"`, i.e. the year is displayed with 4 digits.

It should be noted that this scheme differs from the R approach in several ways. First, R writes day names for some time ranges, in a convention that is seldom seen in the literature. Second, R will write nn:mm for both HH:MM and MM:SS, an ambiguity that might confuse readers. Third, the use of both large and small tick marks is not something that R does.

Bear in mind that `tformat` may be set to alter the number format, but that the tick mark scheme cannot (presently) be controlled.

Value

A vector of times corresponding to axis ticks is returned silently.

Author(s)

Dan Kelley

See Also

This is used mainly by `oce.plot.ts`.

`oce.colorsGebco` *Gebco Colors*

Description

Gebco Colors

Usage

```
oce.colorsGebco(n = 9, region = c("water", "land", "both"),
  type = c("fill", "line"))
```

Arguments

`n`	Number of colors to return
`region`	String indicating application region, one of `"water"`, `"land"`, or `"both"`.
`type`	String indicating the purpose, one of `"fill"` or `"line"`.

See Also

Other things related to colors: `colormap`, `colors`, `oce.colorsTwo`

`oce.colorsTwo` *Create a Palette of Colours*

Description

Create a palette of colours.

Usage

```
oce.colorsTwo(n, low = 2/3, high = 0, smax = 1, alpha = 1)
```

Arguments

`n`	the number of colours (≥ 1) to be in the palette.
`low`	the hue, in [0, 1], for the low end of a `oce.colorsTwo` scale.
`high`	the hue, in [0, 1], for the high end of a `oce.colorsTwo` scale.
`smax`	the maximum saturation, in [0, 1], for the colours of `oce.colorsTwo`.
`alpha`	the alpha value, in [0, 1], for the colours of `oce.colorsTwo`.

Details

oce.colorsPalette provides a variety of pre-defined palettes. which=1 yields the ColorBrewer diverging red/blue scheme while which=2 yields the ColorBrewer diverging RYB scheme [1].

A family of nine-colour schemes is as follows: which="jet" (or which="9A" or which=9.01 for the Jet scheme; which="9B" or which=9.02 for a scheme similar to Jet but omitting the green, and somewhat desaturating the yellow and cyan.

oce.colorsGebco provides palettes that mimic the GEBCO atlas colours, with shades of blue for water and of brown for land. The blue values go from dark to light, and the brown ones from light to dark; in this way, topographic images have light values near sea-level, and get darker in either deeper water or higher terrain.

oce.colorsJet provides a palette similar to the Matlab "jet" palette.

oce.colorsTwo provides a two-tone palette that fades to white at central values.

oce.colorsViridis provides a matplotlib (python) colour scheme that became the standard in 2015; see [2]. This is a blue-yellow transition that is designed to reproduce well in black-and-white, and also to be interpretable by those with certain forms of colour blindness [3, 4, 5].

oce.colorsCDOM, oce.colorsChlorophyll, oce.colorsDensity, oce.colorsFreesurface, oce.colorsOxygen, oce.colorspAR, oce.colorsPhase, oce.colorsSalinity, oce.colorsTemperature, oce.colorsTurbidity, oce.colorsVelocity and oce.colorsVorticity are based on RGB values set up by Kristen M. Thyng for her Python package named cmcolor [7].

Author(s)

Dan Kelley

References

[1] Color Brewer. http://colorbrewer2.org/

[2] A blog item on the Viridis (and related) matplotlib colour scales is at http://bids.github.io/colormap/.

[3] Light, A., and P. J. Bartlein, 2004. The End of the Rainbow? Color Schemes for Improved Data Graphics. *Eos Trans. AGU*, 85(40), doi:10.1029/2004EO400002.

[4] Martin Jakobsson, Ron Macnab, and Members of the Editorial Board, IBCAO. Selective comparisons of GEBCO (1979) and IBCAO (2000) maps. 'https://www.ngdc.noaa.gov/mgg/bathymetry/arctic/ibcao_ge

[5] Stephenson, David B., 2005. Comment on "Color schemes for improved data graphics," by A. Light and P. J. Bartlein. *Eos Trans. AGU*, 86(20).

[6] The Geography department at the University of Oregon has good resources on colour schemes; see e.g. http://geography.uoregon.edu/datagraphics/color_scales.htm (This URL worked prior to December 8, 2015, but was found to fail on that date; it is included here in case users want to search for themselves.)

[7] The cmocean Python package, written by Kristen M Thyng, is available at https://github.com/kthyng/cmocean.

See Also

Other things related to colors: colormap, colors, oce.colorsGebco

Examples

```
library(oce)
opar <- par(no.readonly = TRUE)
# 1. Show a few palettes
x <- array(1:1000, dim=c(1, 1000))
par(mfrow=c(1, 5), mar=c(1, 3, 3, 1))
image(x, col=oce.colorsTwo(200), main="oce.colorsTwo")
image(x, col=oce.colorsJet(200), main="oce.colorsJet")
image(x, col=oce.colorsGebco(200), main="oce.colorsGebco")
image(x, col=oce.colorsPalette(200), main="oce.colorsPalette")
image(x, col=oce.colorsViridis(200), main="oce.colorsViridis")

# 4. Kristen M Thyng's 'cmocean' colours, specialised for oceanography.
par(mfrow=c(3, 4), mar=c(1, 3, 3, 1))
image(x, col=oce.colorsCDOM(200), main="oce.colorsCDOM")
image(x, col=oce.colorsChlorophyll(200), main="oce.colorsChlorophyll")
image(x, col=oce.colorsDensity(200), main="oce.colorsDensity")
image(x, col=oce.colorsFreesurface(200), main="oce.colorsFreesurface")
image(x, col=oce.colorsOxygen(200), main="oce.colorsOxygen")
image(x, col=oce.colorsPAR(200), main="oce.colorsPAR")
image(x, col=oce.colorsPhase(200), main="oce.colorsPhase")
image(x, col=oce.colorsSalinity(200), main="oce.colorsSalinity")
image(x, col=oce.colorsTemperature(200), main="oce.colorsTemperature")
image(x, col=oce.colorsTurbidity(200), main="oce.colorsTurbidity")
image(x, col=oce.colorsVelocity(200), main="oce.colorsVelocity")
image(x, col=oce.colorsVorticity(200), main="oce.colorsVorticity")

# 3. Acoustic-Doppler profiler data; note that plot,adp-method() puts makes
# zlim be symmetric about zero velocity.
par(mfrow=c(1, 1))
data(adp)
plot(adp, which='u1')

# 4. Contrast Jet with Viridis, using standard Volcano dataset;
# try printing the results in black and white.
par(mfrow=c(2, 1))
imagep(volcano, col=oce.colorsJet)
imagep(volcano, col=oce.colorsViridis)
```

`oce.contour` *Oce Variant of contour*

Description

This provides something analagous to contour, but with the ability to flip x and y. Setting `revy=TRUE` can be helpful if the `y` data represent pressure or depth below the surface.

Usage

```
oce.contour(x, y, z, revx = FALSE, revy = FALSE, add = FALSE, tformat,
  drawTimeRange = getOption("oceDrawTimeRange"),
  debug = getOption("oceDebug"), ...)
```

Arguments

`x`	values for x grid.
`y`	values for y grid.
`z`	matrix for values to be contoured. The first dimension of z must equal the number of items in x, etc.
`revx`	set to TRUE to reverse the order in which the labels on the x axis are drawn
`revy`	set to TRUE to reverse the order in which the labels on the y axis are drawn
`add`	logical value indicating whether the contours should be added to a pre-existing plot.
`tformat`	time format; if not supplied, a reasonable choice will be made by `oce.axis.POSIXct`, which draws time axes.
`drawTimeRange`	logical, only used if the x axis is a time. If TRUE, then an indication of the time range of the data (not the axis) is indicated at the top-left margin of the graph. This is useful because the labels on time axes only indicate hours if the range is less than a day, etc.
`debug`	a flag that turns on debugging; set to 1 to information about the processing.
`...`	optional arguments passed to plotting functions.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(topoWorld)
## coastline now, and in last glacial maximum
lon <- topoWorld[["longitude"]]
lat <- topoWorld[["latitude"]]
z <- topoWorld[["z"]]
oce.contour(lon, lat, z, levels=0, drawlabels=FALSE)
oce.contour(lon, lat, z, levels=-130, drawlabels=FALSE, col='blue', add=TRUE)
```

oce.grid *Add a Grid to an Existing Oce Plot*

Description

Add a Grid to an Existing Oce Plot

Usage

```
oce.grid(xat, yat, col = "lightgray", lty = "dotted", lwd = par("lwd"))
```

Arguments

xat	either a list of x values at which to draw the grid, or the return value from an oce plotting function
yat	a list of y values at which to plot the grid (ignored if gx was a return value from an oce plotting function)
col	colour of grid lines (see par)
lty	type for grid lines (see par)
lwd	width for grid lines (see par)

Details

For plots not created by oce functions, or for missing xat and yat, this is the same as a call to grid with missing nx and ny. However, if xat is the return value from certain oce functions, a more sophisticated grid is constructed. The problem with grid is that it cannot handle axes with non-uniform grids, e.g. those with time axes that span months of differing lengths.

As of early February 2015, oce.grid handles xat produced as the return value from the following functions: imagep and oce.plot.ts, plot,adp-method, plot,echosounder-method, and plotTS. It makes no sense to try to use oce.grid for multipanel oce plots, e.g. the default plot from plot,adp-method.

Examples

```
library(oce)
i <- imagep(volcano)
oce.grid(i, lwd=2)

data(sealevel)
i <- oce.plot.ts(sealevel[["time"]], sealevel[["elevation"]])
oce.grid(i, col='red')

data(ctd)
i <- plotTS(ctd)
oce.grid(i, col='red')

data(adp)
```

```
i <- plot(adp, which=1)
oce.grid(i, col='gray', lty=1)

data(echosounder)
i <- plot(echosounder)
oce.grid(i, col='pink', lty=1)
```

---

oce.plot.ts *Oce Variant of plot.ts*

Description

Plot a time-series, obeying the timezone and possibly drawing the range in the top-left margin.

Usage

```
oce.plot.ts(x, y, type = "l", xlim, ylim, xlab, ylab, drawTimeRange,
  fill = FALSE, xaxs = par("xaxs"), yaxs = par("yaxs"),
  cex = par("cex"), cex.axis = par("cex.axis"),
  cex.main = par("cex.main"), mgp = getOption("oceMgp"), mar = c(mgp[1] +
  if (nchar(xlab) > 0) 1.5 else 1, mgp[1] + 1.5, mgp[2] + 1, mgp[2] + 3/4),
  main = "", despike = FALSE, axes = TRUE, tformat,
  marginsAsImage = FALSE, grid = FALSE, grid.col = "darkgray",
  grid.lty = "dotted", grid.lwd = 1, debug = getOption("oceDebug"), ...)
```

Arguments

x	the times of observations.
y	the observations.
type	plot type, "l" for lines, "p" for points.
xlim	optional limit for x axis. This has an additional effect, beyond that for conventional R functions: it effectively windows the data, so that autoscaling will yield limits for y that make sense within the window.
ylim	optional limit for y axis.
xlab	name for x axis; defaults to "".
ylab	name for y axis; defaults to the plotted item.
drawTimeRange	an optional indication of whether/how to draw a time range, in the top-left margin of the plot; see oce.axis.POSIXct for details.
fill	boolean, set TRUE to fill the curve to zero (which it does incorrectly if there are missing values in y).
xaxs	control x axis ending; see par("xaxs").
yaxs	control y axis ending; see par("yaxs").
cex	size of labels on axes; see par("cex").

`cex.axis`	see par("cex.axis").
`cex.main`	see par("cex.main").
`mgp`	3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
`mar`	value to be used with `par("mar")` to set margins. The default value uses significantly tighter margins than is the norm in R, which gives more space for the data. However, in doing this, the existing `par("mar")` value is ignored, which contradicts values that may have been set by a previous call to `drawPalette`. To get plot with a palette, first call `drawPalette`, then call `oce.plot.ts` with `mar=par("mar")`.
`main`	title of plot.
`despike`	boolean flag that can turn on despiking with `despike`.
`axes`	boolean, set to `TRUE` to get axes plotted
`tformat`	optional format for labels on the time axis
`marginsAsImage`	boolean indicating whether to set the right-hand margin to the width normally taken by an image drawn with `imagep`.
`grid`	if `TRUE`, a grid will be drawn for each panel. (This argument is needed, because calling `grid` after doing a sequence of plots will not result in useful results for the individual panels.
`grid.col`	colour of grid
`grid.lty`	line type of grid
`grid.lwd`	line width of grid
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	graphical parameters passed down to `plot`.

Details

Depending on the version of R, the standard `plot` and `plot.ts` routines will not obey the time zone of the data. This routine gets around that problem. It can also plot the time range in the top-left margin, if desired; this string includes the timezone, to remove any possible confusion. The time axis is drawn with `oce.axis.POSIXct`.

Value

A list is silently returned, containing `xat` and `yat`, values that can be used by `oce.grid` to add a grid to the plot.

Author(s)

Dan Kelley

Examples

```
library(oce)
t0 <- as.POSIXct("2008-01-01", tz="UTC")
t <- seq(t0, length.out=48, by="30 min")
y <- sin(as.numeric(t - t0) * 2 * pi / (12 * 3600))
oce.plot.ts(t, y, type='l', xaxs='i')
```

`oce.write.table` *Write the Data Portion of Object to a File*

Description

The output has a line containing the names of the columns in `x$data`, each enclosed in double quotes. After that line are lines for the data themselves. The default is to separate data items by a single space character, but this can be altered by using a `sep` argument in the `...` list (see `write.table`).

Usage

```
oce.write.table(x, file = "", ...)
```

Arguments

`x`	an oce object that contains a `data` table.
`file`	file name, as passed to `write.table`. Use `""` to get a listing in the terminal window.
`...`	optional arguments passed to `write.table`.

Details

This function is little more than a thin wrapper around `write.table`, the only difference being that row names are omitted here, making for a file format that is more conventional in Oceanography.

Value

The value of `write.table` is returned.

Author(s)

Dan Kelley

See Also

`write.table`, which does the actual work.

`oceApprox` *Interpolate 1D Data with UNESCO or Reiniger-Ross Algorithm*

Description

Interpolate one-dimensional data using schemes that permit curvature but tends minimize extrema that are not well-indicated by the data.

Usage

```
oceApprox(x, y, xout, method = c("rr", "unesco"))
```

Arguments

`x`	the independent variable (z or p, usually).
`y`	the dependent variable.
`xout`	the values of the independent variable at which interpolation is to be done.
`method`	method to use. See "Details".

Details

Setting `method="rr"` yields the weighted-parabola algorithm of Reiniger and Ross (1968). For procedure is as follows. First, the interpolant for any `xout` value that is outside the range of `x` is set to NA. Next, linear interpolation is used for any `xout` value that has only one smaller neighboring `x` value, or one larger neighboring value. For all other values of `xout`, the 4 neighboring points `x` are sought, two smaller and two larger. Then two parabolas are determined, one from the two smaller points plus the nearest larger point, and the other from the nearest smaller point and the two larger points. A weighted sum of these two parabolas provides the interpolated value. Note that, in the notation of Reiniger and Ross (1968), this algorithm uses `m=2` and `n=1`. (A future version of this routine might provide the ability to modify these values.)

Setting `method="unesco"` yields the method that is used by the U.S. National Oceanographic Data Center. It is described in pages 48-50 of reference 2; reference 3 presumably contains the same information but it is not as easily accessible. The method works as follows.

- If there are data above 5m depth, then the surface value is taken to equal to the shallowest recorded value.
- Distance bounds are put on the four neighboring points, and the Reiniger-Ross method is used for interpolated points with sufficiently four close neighbors. The bounds are described in table 15 of reference 2 only for so-called standard depths; in the present instance they are transformed to the following rules. Inner neighbors must be within 5m for data above 10m, 50m above 250m 100m above 900m, 200m above 2000m, or within 1000m otherwise. Outer neighbors must be within 200m above 500m, 400m above 1300m, or 1000m otherwise. If two or more points meet these criteria, Lagrangian interpolation is used. If not, `NA` is used as the interpolant.

After these rules are applied, the interpolated value is compared with the values immediately above and below it, and if it is outside the range, simple linear interpolation is used.

Value

A vector of interpolated values, corresponding to the `xout` values and equal in number.

Author(s)

Dan Kelley

References

1. R.F. Reiniger and C.K. Ross, 1968. A method of interpolation with application to oceanographic data. *Deep Sea Research*, **15**, 185-193.
2. Daphne R. Johnson, Tim P. Boyer, Hernan E. Garcia, Ricardo A. Locarnini, Olga K. Baranova, and Melissa M. Zweng, 2011. World Ocean Database 2009 Documentation. NODC Internal report 20. Ocean Climate Laboratory, National Oceanographic Data Center. Silver Spring, Maryland.
3. UNESCO, 1991. Processing of oceanographic station data, 138 pp., Imprimerie des Presses Universitaires de France, United Nations Educational, Scientific and Cultural Organization, France.

Examples

```
library(oce)
if (require(ocedata)) {
    data(RRprofile)
    zz <- seq(0, 2000, 2)
    plot(RRprofile$temperature, RRprofile$depth, ylim=c(500, 0), xlim=c(2, 11))
    ## Contrast two methods
    a1 <- oce.approx(RRprofile$depth, RRprofile$temperature, zz, "rr")
    a2 <- oce.approx(RRprofile$depth, RRprofile$temperature, zz, "unesco")
    lines(a1, zz)
    lines(a2, zz, col='red')
    legend("bottomright",lwd=1,col=1:2, legend=c("rr","unesco"),cex=3/4)
}
```

`oceConvolve` *Convolve two time series*

Description

Convolve two time series, using a backward-looking method. This function provides a straightforward convolution, which may be useful to those who prefer not to use `convolve` and `filter` in the `stats` package.

Usage

```
oceConvolve(x, f, end = 2)
```

Arguments

x	a numerical vector of observations.
f	a numerical vector of filter coefficients.
end	a flag that controls how to handle the points of the x series that have indices less than the length of f. If end=0, the values are set to 0. If end=1, the original x values are used there. If end=2, that fraction of the f values that overlap with x are used.

Value

A vector of the convolution output.

Author(s)

Dan Kelley

Examples

```
library(oce)
t <- 0:1027
n <- length(t)
signal <- ifelse(sin(t * 2 * pi / 128) > 0, 1, 0)
tau <- 10
filter <- exp(-seq(5*tau, 0) / tau)
filter <- filter / sum(filter)
observation <- oce.convolve(signal, filter)
plot(t, signal, type='l')
lines(t, observation, lty='dotted')
```

---

oceCRS *Coordinate Reference System strings for some oceans*

Description

Create a coordinate reference string (CRS), suitable for use as a `projection` argument to `mapPlot` or `plot,coastline-method`.

Usage

```
oceCRS(region)
```

Arguments

region	character string indicating the region. This must be in the following list (or a string that matches to just one entry, with `pmatch`): "North Atlantic", "South Atlantic", "Atlantic", "North Pacific", "South Pacific", "Pacific", "Arctic", and "Antarctic".

Value

string contain a CRS, which can be used as `projection` in `mapPlot`.

Caution

This is a preliminary version of this function, with the results being very likely to change through the autumn of 2016, guided by real-world usage.

Author(s)

Dan Kelley

See Also

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `shiftLongitude`, `usrLonLat`, `utm2lonlat`

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
par(mar=c(2, 2, 1, 1))
plot(coastlineWorld, proj=oceCRS("Atlantic"), span=12000)
plot(coastlineWorld, proj=oceCRS("North Atlantic"), span=8000)
plot(coastlineWorld, proj=oceCRS("South Atlantic"), span=8000)
plot(coastlineWorld, proj=oceCRS("Arctic"), span=4000)
plot(coastlineWorld, proj=oceCRS("Antarctic"), span=10000)
# Avoid ugly horizontal lines, an artifact of longitude shifting.
# Note: we cannot fill the land once we shift, either.
pacific <- coastlineCut(coastlineWorld, -180)
plot(pacific, proj=oceCRS("Pacific"), span=15000, col=NULL)
plot(pacific, proj=oceCRS("North Pacific"), span=12000, col=NULL)
plot(pacific, proj=oceCRS("South Pacific"), span=12000, col=NULL)

## End(Not run)
```

`oceDebug` *Print a debugging message*

Description

Print an indented debugging message. Many oce functions decrease the debug level by 1 when they call other functions, so the effect is a nesting, with more space for deeper function level.

Usage

```
oceDebug(debug = 0, ..., unindent = 0)
```

Arguments

debug	an integer, less than or equal to zero for no message, and greater than zero for increasing levels of debugging. Values greater than 4 are treated like 4.
...	items to be supplied to cat, which does the printing. Almost always, this should include a trailing newline.
unindent	Number of levels to un-indent, e.g. for start and end lines from a called function.

Author(s)

Dan Kelley

Examples

```
foo <- function(debug)
{
   oceDebug(debug, "in function foo\n")
}
debug <- 1
oceDebug(debug, "in main")
foo(debug=debug-1)
```

---

oceDeleteData   *Delete data from an* oce *object*

Description

Return a copy of the supplied object that lacks the named element in its data slot, and that has a note about the deletion in its processing log.

Usage

```
oceDeleteData(object, name)
```

Arguments

object	an oce object
name	String indicating the name of the item to be deleted.

oceDeleteMetadata — *Delete metadata from an* oce *object*

Description

Return a copy of the supplied object that lacks the named element in its `metadata` slot, and that has a note about the deletion in its processing log.

Usage

```
oceDeleteMetadata(object, name)
```

Arguments

`object`	an oce object
`name`	String indicating the name of the item to be deleted.

oceEdit — *Edit an Oce Object*

Description

Edit an element of an oce object, inserting a note in the processing log of the returned object.

Usage

```
oceEdit(x, item, value, action, reason = "", person = "",
  debug = getOption("oceDebug"))
```

Arguments

`x`	an `oce` object. The exact action of `oceEdit` depends on the `class` of `x`.
`item`	if supplied, a character string naming an item in the object's `metadata` or `data` slot, the former being checked first. An exception is if `item` starts with `"data@"` or `"metadata@"`, in which case the named slot is updated with a changed value of the contents of `item` after the `@` character.
`value`	new value for `item`, if both supplied.
`action`	optional character string containing R code to carry out some action on the object.
`reason`	character string giving the reason for the change.
`person`	character string giving the name of person making the change.
`debug`	an integer that specifies a level of debugging, with 0 or less indicating no debugging, and 1 or more indicating debugging.

Details

There are several ways to use this function.

- Case 1. If both an `item` and `value` are supplied, then either the object's metadata or data slot may be altered. There are two ways in which this can be done.
  - Case 1A. If the `item` string does not contain an `@` character, then the `metadata` slot is examined for an entry named `item`, and that is modified if so. Alternatively, if `item` is found in `metadata`, then that value is modified. However, if `item` is not found in either `metadata` or `data`, then an error is reported (see 1B for how to add something that does not yet exist).
  - Case 1B. If the `item` string contains the `@` character, then the text to the left of that character must be either `"metadata"` or `"data"`, and it names the slot in which the change is done. In contrast with case 1A, this will *create* a new item, if it is not already in existence.
- Case 2. If `item` and `value` are not supplied, then `action` must be supplied. This is a character string specifying some action to be performed on the object, e.g. a manipulation of a column. The action must refer to the object as `x`; see Examples.

In any case, a log entry is stored in the object, to document the change. Indeed, this is the main benefit to using this function, instead of altering the object directly. The log entry will be most useful if it contains a brief note on the `reason` for the change, and the name of the `person` doing the work.

Value

An object of class `"oce"`, altered appropriately, and with a log item indicating the nature of the alteration.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
ctd2 <- oceEdit(ctd, item="latitude", value=47.8879,
                reason="illustration", person="Dan Kelley")
ctd3 <- oceEdit(ctd,action="x@data$pressure<-x@data$pressure-1")
```

`oceFilter` *Filter a time-series*

Description

Filter a time-series, possibly recursively

Usage

```
oceFilter(x, a = 1, b, zero.phase = FALSE)
```

Arguments

x	a vector of numeric values, to be filtered as a time series.
a	a vector of numeric values, giving the a coefficients (see "Details").
b	a vector of numeric values, giving the b coefficients (see "Details").
zero.phase	boolean, set to TRUE to run the filter forwards, and then backwards, thus removing any phase shifts associated with the filter.

Details

The filter is defined as e.g. $y[i] = b[1]*x[i]+b[2]*x[i-1]+b[3]*x[i-2]+...-a[2]*y[i-1]-a[3]*y[i-2]-a[4]*y[i-3]-...$, where some of the illustrated terms will be omitted if the lengths of a and b are too small, and terms are dropped at the start of the time series where the index on x would be less than 1.

By contrast with the filter function of R, oce.filter lacks the option to do a circular filter. As a consequence, oce.filter introduces a phase lag. One way to remove this lag is to run the filter forwards and then backwards, as in the "Examples". However, the result is still problematic, in the sense that applying it in the reverse order would yield a different result. (Matlab's filtfilt shares this problem.)

Value

A numeric vector of the filtered results, y, as denoted in "Details".

Note

The first value in the a vector is ignored, and if length(a) equals 1, a non-recursive filter results.

Author(s)

Dan Kelley

Examples

```
library(oce)
par(mar=c(4, 4, 1, 1))
b <- rep(1, 5)/5
a <- 1
x <- seq(0, 10)
y <- ifelse(x == 5, 1, 0)
f1 <- oce.filter(y, a, b)
plot(x, y, ylim=c(-0, 1.5), pch="o", type='b')
points(x, f1, pch="x", col="red")

# remove the phase lag
```

```
f2 <- oce.filter(y, a, b, TRUE)
points(x, f2, pch="+", col="blue")

legend("topleft", col=c("black","red","blue"), pch=c("o","x","+"),
       legend=c("data","normal filter", "zero-phase filter"))
mtext("note that normal filter rolls off at end")
```

---

oceGetData *Get data from an* `oce` *object*

Description

In contrast to the various `[[` functions, this is guaranteed to look only within the `data` slot. If the named item is not found, `NULL` is returned.

Usage

```
oceGetData(object, name)
```

Arguments

`object`	an oce object
`name`	String indicating the name of the item to be found.

---

oceGetMetadata *Get metadata element from an* `oce` *object*

Description

In contrast to the various `[[` functions, this is guaranteed to look only within the `metadata` slot. If the named item is not found, `NULL` is returned.

Usage

```
oceGetMetadata(object, name)
```

Arguments

`object`	an oce object
`name`	String indicating the name of the item to be found.

oceMagic *Find the Type of an Oceanographic Data File*

Description

`oceMagic` tries to infer the file type, based on the data within the file, the file name, or a combination of the two.

Usage

```
oceMagic(file, debug = getOption("oceDebug"))
```

Arguments

`file`	a connection or a character string giving the name of the file to be checked.
`debug`	an integer, set non-zero to turn on debugging. Higher values indicate more debugging.

Details

`oceMagic` was previously called `oce.magic`, but that alias will be removed towards the end of the year 2016; see oce-deprecated.

Value

A character string indicating the file type, or `"unknown"`, if the type cannot be determined. If the result contains `"/"` characters, these separate a list describing the file type, with the first element being the general type, the second element being the manufacturer, and the third element being the manufacturer's name for the instrument. For example, `"adp/nortek/aquadopp"` indicates a acoustic-doppler profiler made by NorTek, of the model type called AquaDopp.

Author(s)

Dan Kelley

See Also

This is used mainly by `read.oce`.

oceNames2whpNames *Translate Oce Data Names to WHP Data Names*

Description

Translate oce-style names to WOCE names, using gsub to match patterns. For example, the pattern "oxygen" is taken to mean "CTDOXY".

Usage

```
oceNames2whpNames(names)
```

Arguments

names	vector of strings holding oce-style names.

Value

vector of strings holding WHP-style names.

Author(s)

Dan Kelley

References

Several online sources list WHP names. An example is https://geo.h2o.ucsd.edu/documentation/manuals/pdf/90_1/chap4.pdf

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Other functions that interpret variable names and units from headers: ODFNames2oceNames, cnvName2oceName, oceUnits2whpUnits, unitFromStringRsk, unitFromString, woceNames2oceNames, woceUnit2oceUnit

ocePmatch *Partial matching of strings or numbers*

Description

An extended version of pmatch that allows x to be numeric or string-based. As with pmatch, partial string matches are handled. This is a wrapper that is useful mainly for which arguments to plotting functions.

Usage

```
ocePmatch(x, table, nomatch = NA_integer_, duplicates.ok = FALSE)
```

Arguments

x	a code, or vector of codes. This may be numeric, in which case it is simply returned without further analysis of the other arguments, or it may be string-based, in which case pmatch is used to find numeric matches.
table	a list that maps strings to numbers; pmatch is used on names(table). If the name contains characters that are normally not permitted in a variable name, use quotes, e.g. list(salinity=1, temperature=2, "salinity+temperature"=3).
nomatch	value to be returned for cases of no match (passed to pmatch.
duplicates.ok	code for the handling of duplicates (passed to pmatch).

Value

A number, or vector of numbers, corresponding to the matches. Non-matches are indicated with NA values, or whatever value is given by the NA argument.

Author(s)

Dan Kelley

See Also

Since pmatch is used for the actual matching, its documentation should be consulted.

Examples

```
library(oce)
oce.pmatch(c("s", "at", "te"), list(salinity=1, temperature=3.1))
```

oceSetData	*Set something in the* data *slot of an* oce *object*

Description

Set something in the data slot of an oce object

Usage

```
oceSetData(object, name, value, unit, originalName, note = "")
```

Arguments

object	an oce object
name	String indicating the name of the item to be set.
value	Value for the item.
unit	An optional indication of the units for the item. This has three possible forms (see "Details").
originalName	Optional character string giving an 'original' name (e.g. as stored in the header of a data file).
note	A note to be stored in the processing log. If an empty string (the default) then an entry will be constructed from the function call. If NULL, then no entry will be added to the processing log.

Details

There are three possibilities for unit:

- *Case 1.* unit is a named or unnamed list that contains two items. If the list is named, the names must be unit and scale. If the list is unnamed, the stated names are assigned to the items, in the stated order. Either way, the unit item must be an expression that specifies the unit, and the scale item must be a string that describes the scale. For example, modern temperatures have unit=list(unit=expression(degree*C), scale="ITS-90").
- *Case 2.* unit is an expression giving the unit as above. In this case, the scale will be set to "".
- *Case 3.* unit is a character string that is converted into an expression with parse(text=unit), and the scale set to "".

Examples

```
data(ctd)
Tf <- swTFreeze(ctd)
ctd <- oceSetData(ctd, "freezing", Tf, list(unit=expression(degree*C), scale="ITS-90"))
feet <- swDepth(ctd) / 0.3048
ctd <- oceSetData(ctd, name="depthInFeet", value=feet, expression("feet"))
fathoms <- feet / 6
ctd <- oceSetData(ctd, "depthInFathoms", fathoms, "fathoms")
```

oceSetMetadata — *Set something in the `metadata` slot of an `oce` object*

Description

Set something in the `metadata` slot of an `oce` object

Usage

```
oceSetMetadata(object, name, value, note = "")
```

Arguments

`object`	an `oce` object
`name`	String indicating the name of the item to be set.
`value`	Value for the item.
`note`	A note to be stored in the processing log.

oceSmooth — *Smooth an Oce Object*

Description

Each data element is smoothed as a timeseries. For ADP data, this is done along time, not distance. Time vectors, if any, are not smoothed. A good use of `oce.smooth` is for despiking noisy data.

Usage

```
oceSmooth(x, ...)
```

Arguments

`x`	an `oce` object.
`...`	parameters to be supplied to `smooth`, which does the actual work.

Value

An object of `class` `"oce"` that has been smoothed appropriately.

Author(s)

Dan Kelley

See Also

The work is done with `smooth`, and the `...` arguments are handed to it directly by `oce.smooth`.

Examples

```
library(oce)
data(ctd)
d <- oce.smooth(ctd)
plot(d)
```

`oceSpectrum` *Wrapper to give normalized spectrum*

Description

This is a wrapper around the R `spectrum` function, which returns spectral values that are adjusted so that the integral of those values equals the variance of the input `x`.

Usage

```
oceSpectrum(x, ...)
```

Arguments

`x`	As for `spectrum`, a univariate or multivariate time series.
`...`	extra arguments passed on to `spectrum`.

Value

A spectrum that has values that integrate to the variance.

Author(s)

Dan Kelley

See Also

`spectrum`.

Examples

```
x <- rnorm(1e3)
s <- spectrum(x, plot=FALSE)
ss <- oce.spectrum(x, plot=FALSE)
cat("variance of x=", var(x), "\n")
cat("integral of     spectrum=", sum(s$spec)*diff(s$freq[1:2]), "\n")
cat("integral of oce.spectrum=", sum(ss$spec)*diff(ss$freq[1:2]), "\n")
```

oceUnits2whpUnits *Translate oce unit to WHP unit*

Description

Translate oce units to WHP-style strings, to match patterns. For example, the pattern "oxygen" is taken to mean "CTDOXY".

Usage

```
oceUnits2whpUnits(units, scales)
```

Arguments

units	vector of expressions for units in oce notation.
scales	vector of strings for scales in oce notation.

Value

vector of strings holding WOCE-style names.

Author(s)

Dan Kelley

References

Several online sources list WOCE names. An example is https://geo.h2o.ucsd.edu/documentation/manuals/pdf/90_1/chap4.pdf

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Other functions that interpret variable names and units from headers: ODFNames2oceNames, cnvName2oceName, oceNames2whpNames, unitFromStringRsk, unitFromString, woceNames2oceNames, woceUnit2oceUnit odf-class *Class to Store ODF data*

Description

Class for data stored in a format used at Canadian Department of Fisheries and Oceans laboratories. This is somewhat unusual amongst oce classes, in that it does not map to a particular instrument, but rather to a storage type; in that sense, it is similar to the bremen-class.

Methods

Consider an ODF object named odf.

Accessing metadata.

Metadata (contained in the S4 slot named metadata) may be retrieved or set by name, odf[["longitude"]] <- odf[["long
corrects a one-degree error.

Accessing measured data.

Column data may be accessed by name, e.g. odf[["salinity"]], odf[["temperature"]], odf[["pressure"]], etc. It is up to the user to realize what is in the object.

Assigning values.

Items stored in the object may be altered with e.g. odf[["salinity"]] <- rep(35,10).

Overview of contents.

The show method (e.g. show(odf)) displays information about the object.

Author(s)

Dan Kelley

See Also

Other things related to odf data: ODF2oce, ODFNames2oceNames, [[,odf-method, [[<-,odf-method, plot,odf-method, read.ctd.odf, read.odf, subset,odf-method, summary,odf-method

Other classes provided by oce: adp-class, adv-class, argo-class, bremen-class, cm-class, coastline-class, ctd-class, echosounder-class, lisst-class, lobo-class, met-class, oce-class, rsk-class, sealevel-class, section-class, topo-class, windrose-class

ODF2oce *Create ODF object from the output of* `ODF::read_ODF()`

Description

As of August 11, 2015, `ODF::read_ODF` returns a list with 9 elements, one named `DATA`, which is a `data.frame` containing the columnar data, the others being headers of various sorts. The present function constructs an oce object from such data, facilitating processing and plotting with the general oce functions. This involves storing the 8 headers verbatim in the `odfHeaders` in the `metadata` slot, and also copying some of the header information into more standard names (e.g. `metadata@longitude` is a copy of `metadata@odfHeader$EVENT_HEADER$INITIAL_LATITUDE`). As for the `DATA`, they are stored in the `data` slot, after renaming from ODF to oce convention using `ODFNames2oceNames`.

Usage

```
ODF2oce(ODF, coerce = TRUE, debug = getOption("oceDebug"))
```

Arguments

`ODF`	A list as returned by `read_ODF` in the `ODF` package
`coerce`	A logical value indicating whether to coerce the return value to an appropriate object type, if possible.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Value

An oce object, possibly coerced to a subtype.

Caution

This function may change as the `ODF` package changes. Since `ODF` has not been released yet, this should not affect any users except those involved in the development of oce and `ODF`.

Author(s)

Dan Kelley

See Also

Other things related to odf data: `ODFNames2oceNames`, `[[,odf-method`, `[[<-,odf-method`, `odf-class`, `plot,odf-method`, `read.ctd.odf`, `read.odf`, `subset,odf-method`, `summary,odf-method`

ODFNames2oceNames *Translate from ODF Names to Oce Names*

Description

Translate data names in the ODF convention to similar names in the Oce convention.

Usage

```
ODFNames2oceNames(ODFnames, ODFunits = NULL, columns = NULL,
  PARAMETER_HEADER = NULL, debug = getOption("oceDebug"))
```

Arguments

ODFnames	Vector of strings holding ODF names.
ODFunits	Vector of strings holding ODF units.
columns	Optional list containing name correspondances, as described for read.ctd.odf.
PARAMETER_HEADER	Optional list containing information on the data variables.
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

The following table gives the regular expressions that define recognized ODF names, along with the translated names as used in oce objects. If an item is repeated, then the second one has a 2 appended at the end, etc. Note that quality-control columns (with names starting with "QQQQ") are not handled with regular expressions. Instead, if such a flag is found in the i-th column, then a name is constructed by taking the name of the (i-1)-th column and appending "Flag".

Regexp	**Result**	**Notes**
ALTB_*.*	altimeter	
BATH_*.*	barometricDepth	Barometric depth (of sensor? of water column?)
BEAM_*.*	a	Used in adp objects
CNTR_*.*	scan	Used in ctd objects
CRAT_*.*	conductivity	Conductivity ratio
COND_*.*	conductivity	Conductivity in mS/cm or S/m (unit detected)
CNDC_*.*	conductivity	Conductivity in mS/cm or S/m (unit detected)
DEPH_*.*	pressure	Sensor depth below sea level
DOXY_*.*	oxygen	Used mainly in ctd objects
ERRV_*.*	error	Used in adp objects
EWCT_*.*	u	Used in adp and cm objects

FFFF_*.*	flagArchaic	Old flag name, replaced by QCFF
FLOR_*.*	fluorometer	Used mainly in ctd objects
FWETLABS	fwetlabs	Used in ??
HCDM	directionMagnetic	
HCDT	directionTrue	
HCSP	speedHorizontal	
LATD_*.*	latitude	
LOND_*.*	longitude	
NSCT_*.*	v	Used in adp objects
OCUR_*.*	oxygenCurrent	Used mainly in ctd objects
OSAT_*.*	oxygenSaturation	Used mainly in ctd objects
OTMP_*.*	oxygenTemperature	Used mainly in ctd objects
OXYV_*.*	oxygenVoltage	Used mainly in ctd objects
PHPH_*.*	pH	
POTM_*.*	theta	Used mainly in ctd objects
PRES_*.*	pressure	Used mainly in ctd objects
PSAL_*.*	salinity	Used mainly in ctd objects
PSAR_*.*	par	Used mainly in ctd objects
QCFF_*.*	flag	Overall flag
SIGP_*.*	sigmaTheta	Used mainly in ctd objects
SIGT_*.*	sigmat	Used mainly in ctd objects
SNCN_*.*	scanCounter	Used mainly in ctd objects
SYTM_*.*	time	Used in many objects
TE90_*.*	temperature	Used mainly in ctd objects
TEMP_*.*	temperature	Used mainly in ctd objects
TOTP_*.*	pressureAbsolute	Used mainly in ctd objects
UNKN_*.*	-	The result is context-dependent
VCSP_*.*	w	Used in adp objects

Any code not shown in the list is transferred to the oce object without renaming, apart from the adjustment of suffix numbers. The following code have been seen in data files from the Bedford Institute of Oceanography: ALTB, PHPH and QCFF.

Value

A vector of strings.

A note on unit conventions

Some older ODF files contain non-standard units for conductivity, including mho/m, mmho/cm, and mmHo. As the units for conductivity are important for derived quantities (e.g. salinity), such units are converted to standard units (e.g. S/m and mS/cm), with a warning.

Consistency warning

There are disagreements on variable names. For example, the "DFO Common Data Dictionary" [1] has unit millmole/m^3 for NODC and MEDS, but it has unit mL/L for BIO and IML.

Author(s)

Dan Kelley

References

1. The Department of Fisheries and Oceans Common Data Dictionary may be available at `http://www.isdm.gc.ca/isdm-g` although that link seems to be unreliable. As of September 2017, the link `https://slgo.ca/app-sgdo/en/docs_reference/format_odf.html` seems to be a good place to start.

See Also

Other functions that interpret variable names and units from headers: `cnvName2oceName`, `oceNames2whpNames`, `oceUnits2whpUnits`, `unitFromStringRsk`, `unitFromString`, `woceNames2oceNames`, `woceUnit2oceUnit`

Other things related to odf data: `ODF2oce`, `[[,odf-method`, `[[<-,odf-method`, `odf-class`, `plot,odf-method`, `read.ctd.odf`, `read.odf`, `subset,odf-method`, `summary,odf-method`

`parseLatLon` *Parse a Latitude or Longitude String*

Description

Parse a latitude or longitude string, e.g. as in the header of a CTD file The following formats are understood (for, e.g. latitude)

```
 *
NMEA Latitude = 47 54.760 N ** Latitude: 47 53.27 N
```

Usage

```
parseLatLon(line, debug = getOption("oceDebug"))
```

Arguments

`line`	a character string containing an indication of latitude or longitude.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Value

A numerical value of latitude or longitude.

Author(s)

Dan Kelley

See Also

Used by `read.ctd`.

plot,adp-method *Plot ADP Data*

Description

Create a summary plot of data measured by an acoustic doppler profiler.

Usage

```
## S4 method for signature 'adp'
plot(x, which = 1:dim(x@data$v)[3], mode = c("normal",
  "diagnostic"), col, breaks, zlim, titles, lwd = par("lwd"), type = "l",
  ytype = c("profile", "distance"),
  drawTimeRange = getOption("oceDrawTimeRange"), useSmoothScatter,
  missingColor = "gray", mgp = getOption("oceMgp"), mar = c(mgp[1] + 1.5,
  mgp[1] + 1.5, 1.5, 1.5), mai.palette = rep(0, 4), tformat,
  marginsAsImage = FALSE, cex = par("cex"), cex.axis = par("cex.axis"),
  cex.main = par("cex.main"), xlim, ylim, control, useLayout = FALSE,
  coastline = "coastlineWorld", span = 300, main = "", grid = FALSE,
  grid.col = "darkgray", grid.lty = "dotted", grid.lwd = 1,
  debug = getOption("oceDebug"), ...)
```

Arguments

x	An adp object, i.e. one inheriting from adp-class.
which	list of desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of which.
mode	a string indicating whether to plot the conventional signal (normal) or or, in the special case of Aquadopp single-bin profilers, possibly the diagnostic signal. This argument is ignored except in the case of Aquadopp instruments. Perhaps a third option will become available in the future, for the burst mode that some instruments provide.
col	optional indication of colour(s) to use. If not provided, the default for images is oce.colorsPalette(128,1), and for lines and points is black.
breaks	optional breaks for colour scheme
zlim	a range to be used as the zlim parameter to the imagep call that is used to create the image. If omitted, zlim is set for each panel individually, to encompass the data of the panel and to be centred around zero. If provided as a two-element vector, then that is used for each panel. If provided as a two-column matrix, then each panel of the graph uses the corresponding row of the matrix; for example, setting zlim=rbind(c(-1,1),c(-1,1),c(-.1,.1)) might make sense for which=1:3, so that the two horizontal velocities have one scale, and the smaller vertical velocity has another.

titles
: optional vector of character strings to be used as labels for the plot panels. For images, these strings will be placed in the right hand side of the top margin. For timeseries, these strings are ignored. If this is provided, its length must equal that of `which`.

lwd
: if the plot is of a time-series or scattergraph format with lines, this is used in the usual way; otherwise, e.g. for image formats, this is ignored.

type
: if the plot is of a time-series or scattergraph format, this is used in the usual way, e.g. `"l"` for lines, etc.; otherwise, as for image formats, this is ignored.

ytype
: character string controlling the type of the y axis for images (ignored for time series). If `"distance"`, then the y axis will be distance from the sensor head, with smaller distances nearer the bottom of the graph. If `"profile"`, then this will still be true for upward-looking instruments, but the y axis will be flipped for downward-looking instruments, so that in either case, the top of the graph will represent the sample nearest the sea surface.

drawTimeRange
: boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.

useSmoothScatter
: boolean that indicates whether to use `smoothScatter` in various plots, such as `which="uv"`. If not provided a default is used, with `smoothScatter` being used if there are more than 2000 points to plot.

missingColor
: colour used to indicate `NA` values in images (see `imagep`); set to `NULL` to avoid this indication.

mgp
: A 3-element numerical vector used with `par("mgp")` to control the spacing of axis elements. The default is tighter than the R default.

mar
: A 4-element numerical vector used with `par("mar")` to control the plot margins. The default is tighter than the R default.

mai.palette
: margins, in inches, to be added to those calculated for the palette; alter from the default only with caution

tformat
: optional argument passed to `oce.plot.ts`, for plot types that call that function. (See `strptime` for the format used.)

marginsAsImage
: boolean, `TRUE` to put a wide margin to the right of time-series plots, even if there are no images in the `which` list. (The margin is made wide if there are some images in the sequence.)

cex
: size of labels on axes; see `par`("cex").

cex.axis
: see `par`("cex.axis").

cex.main
: see `par`("cex.main").

xlim
: optional 2-element list for `xlim`, or 2-column matrix, in which case the rows are used, in order, for the panels of the graph.

ylim
: optional 2-element list for `ylim`, or 2-column matrix, in which case the rows are used, in order, for the panels of the graph.

control
: optional list of parameters that may be used for different plot types. Possibilities are `drawBottom` (a boolean that indicates whether to draw the bottom) and `bin` (a numeric giving the index of the bin on which to act, as explained in “Details”).

useLayout	set to FALSE to prevent using layout to set up the plot. This is needed if the call is to be part of a sequence set up by e.g. par(mfrow).
coastline	a coastline object, or a character string naming one. This is used only for which="map". See notes at plot,ctd-method for more information on built-in coastlines.
span	approximate span of map in km
main	main title for plot, used just on the top panel, if there are several panels.
grid	if TRUE, a grid will be drawn for each panel. (This argument is needed, because calling grid after doing a sequence of plots will not result in useful results for the individual panels.
grid.col	colour of grid
grid.lty	line type of grid
grid.lwd	line width of grid
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.
...	optional arguments passed to plotting functions. For example, supplying despike=TRUE will cause time-series panels to be de-spiked with despike. Another common action is to set the colour for missing values on image plots, with the argument missingColor (see imagep). Note that it is an error to give breaks in ..., if the formal argument zlim was also given, because they could contradict each other.

Details

The plot may have one or more panels, with the content being controlled by the which argument.

- which=1:4 (or which="u1" to "u4") yield a distance-time image plot of a velocity component. If x is in beam coordinates (signalled by metadata$oce.coordinate=="beam"), this will be the beam velocity, labelled b[1] etc. If x is in xyz coordinates (sometimes called frame coordinates, or ship coordinates), it will be the velocity component to the right of the frame or ship (labelled u etc). Finally, if x is in "enu" coordinates, the image will show the the eastward component (labelled east). If x is in "other" coordinates, it will be component corresponding to east, after rotation (labelled u\'). Note that the coordinate is set by read.adp, or by beamToXyzAdp, xyzToEnuAdp, or enuToOtherAdp.
- which=5:8 (or which="a1" to "a4") yield distance-time images of backscatter intensity of the respective beams. (For data derived from Teledyn-RDI instruments, this is the item called “echo intensity.”)
- which=9:12 (or which="q1" to "q4") yield distance-time images of signal quality for the respective beams. (For RDI data derived from instruments, this is the item called “correlation magnitude.”)
- which=60 or which="map" draw a map of location(s).

- `which=70:73` (or `which="g1"` to `"g4"`) yield distance-time images of percent-good for the respective beams. (For data derived from Teledyne-RDI instruments, which are the only instruments that yield this item, it is called "percent good.")
- `which=80:83` (or `which="vv"`, `which="va"`, `which="vq"`, and `which="vg"`) yield distance-time images of the vertical beam fields for a 5 beam "SentinelV" ADCP from Teledyne RDI.
- `which="vertical"` yields a two panel distance-time image of vertical beam velocity and amplitude.
- `which=13` (or `which="salinity"`) yields a time-series plot of salinity.
- `which=14` (or `which="temperature"`) yields a time-series plot of temperature.
- `which=15` (or `which="pressure"`) yields a time-series plot of pressure.
- `which=16` (or `which="heading"`) yields a time-series plot of instrument heading.
- `which=17` (or `which="pitch"`) yields a time-series plot of instrument pitch.
- `which=18` (or `which="roll"`) yields a time-series plot of instrument roll.
- `which=19` yields a time-series plot of distance-averaged velocity for beam 1, rightward velocity, eastward velocity, or rotated-eastward velocity, depending on the coordinate system.
- `which=20` yields a time-series of distance-averaged velocity for beam 2, foreward velocity, northward velocity, or rotated-northward velocity, depending on the coordinate system.
- `which=21` yields a time-series of distance-averaged velocity for beam 3, up-frame velocity, upward velocity, or rotated-upward velocity, depending on the coordinate system.
- `which=22` yields a time-series of distance-averaged velocity for beam 4, for `beam` coordinates, or velocity estimate, for other coordinates. (This is ignored for 3-beam data.)
- `which=23` yields a progressive-vector diagram in the horizontal plane, plotted with `asp=1`. Normally, the depth-averaged velocity components are used, but if the `control` list contains an item named `bin`, then the depth bin will be used (with an error resulting if the bin is out of range).
- `which=24` yields a time-averaged profile of the first component of velocity (see `which=19` for the meaning of the component, in various coordinate systems).
- `which=25` as for 24, but the second component.
- `which=26` as for 24, but the third component.
- `which=27` as for 24, but the fourth component (if that makes sense, for the given instrument).
- `which=28` or `"uv"` yields velocity plot in the horizontal plane, i.e. u[2] versus u[1]. If the number of data points is small, a scattergraph is used, but if it is large, `smoothScatter` is used.
- `which=29` or `"uv+ellipse"` as the `"uv"` case, but with an added indication of the tidal ellipse, calculated from the eigen vectors of the covariance matrix.
- `which=30` or `"uv+ellipse+arrow"` as the `"uv+ellipse"` case, but with an added arrow indicating the mean current.
- `which=40` or `"bottomRange"` for average bottom range from all beams of the instrument.
- `which=41` to `44` (or `"bottomRange1"` to `"bottomRange4"`) for bottom range from beams 1 to 4.
- `which=50` or `"bottomVelocity"` for average bottom velocity from all beams of the instrument.

- which=51 to 54 (or "bottomVelocity1" to "bottomVelocity4") for bottom velocity from beams 1 to 4.
- which=55 (or "heaving") for time-integrated, depth-averaged, vertical velocity, i.e. a time series of heaving.
- which=100 (or "soundSpeed") for a time series of sound speed.

In addition to the above, there are some groupings defined:

- which="velocity" equivalent to which=1:3 (velocity components)
- which="amplitude" equivalent to which=5:7 (backscatter intensity components)
- which="quality" equivalent to which=9:11 (quality components)
- which="hydrography" equivalent to which=14:15 (temperature and pressure)
- which="angles" equivalent to which=16:18 (heading, pitch and roll)

The colour scheme for image plots (which in 1:12) is provided by the col argument, which is passed to image to do the actual plotting. See "Examples" for some comparisons.

A common quick-look plot to assess mooring movement is to use which=15:18 (pressure being included to signal the tide, and tidal currents may dislodge a mooring or cause it to settle).

By default, plot,adp-method uses a zlim value for the image that is constructed to contain all the data, but to be symmetric about zero. This is done on a per-panel basis, and the scale is plotted at the top-right corner, along with the name of the variable being plotted. You may also supply zlim as one of the ... arguments, but be aware that a reasonable limit on horizontal velocity components is unlikely to be of much use for the vertical component.

A good first step in the analysis of measurements made from a moored device (stored in d, say) is to do plot(d, which=14:18). This shows time series of water properties and sensor orientation, which is helpful in deciding which data to trim at the start and end of the deployment, because they were measured on the dock or on the ship as it travelled to the mooring site.

Value

A list is silently returned, containing xat and yat, values that can be used by oce.grid to add a grid to the plot.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

Examples

```
library(oce)
data(adp)
plot(adp, which=1:3)
plot(adp, which='temperature', tformat='%H:%M')
```

---

plot,adv-method *Plot ADV data*

Description

Plot ADV data, i.e. stored in an adv-class object.

Usage

```
## S4 method for signature 'adv'
plot(x, which = c(1:3, 14, 15), col, titles, type = "l",
  lwd = par("lwd"), drawTimeRange = getOption("oceDrawTimeRange"),
  drawZeroLine = FALSE, useSmoothScatter, mgp = getOption("oceMgp"),
  mar = c(mgp[1] + 1.5, mgp[1] + 1.5, 1.5, 1.5), tformat,
  marginsAsImage = FALSE, cex = par("cex"), cex.axis = par("cex.axis"),
  cex.main = par("cex.main"), xlim, ylim, brushCorrelation,
  colBrush = "red", main = "", debug = getOption("oceDebug"), ...)
```

Arguments

x	An adv object, i.e. one inheriting from adv-class.
which	List of desired plot types. These are graphed in panels running down from the top of the page. See “Details” for the meanings of various values of which.
col	Optional indication of colour(s) to use. If not provided, the default for images is oce.colorsPalette(128,1), and for lines and points is black.
titles	Optional vector of character strings to be used as labels for the plot panels. For images, these strings will be placed in the right hand side of the top margin. For timeseries, these strings are ignored. If this is provided, its length must equal that of which.
type	Type of plot, as for plot.
lwd	If the plot is of a time-series or scattergraph format with lines, this is used in the usual way; otherwise, e.g. for image formats, this is ignored.
drawTimeRange	Boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.

drawZeroLine	Logical value indicating whether to draw zero lines on velocities.
useSmoothScatter	Logical value indicating whether to use smoothScatter in various plots, such as which="uv". If not provided a default is used, with smoothScatter being used if there are more than 2000 points to plot.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	Value to be used with par("mar").
tformat	Optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
marginsAsImage	Logical value indicating whether to put a wide margin to the right of time-series plots, matching the space used up by a palette in an imagep plot.
cex	Size of labels on axes; see par("cex").
cex.axis	See par("cex.axis").
cex.main	See par("cex.main").
xlim	Optional 2-element list for xlim, or 2-column matrix, in which case the rows are used, in order, for the panels of the graph.
ylim	Optional 2-element list for ylim, or 2-column matrix, in which case the rows are used, in order, for the panels of the graph.
brushCorrelation	Optional number between 0 and 100, indicating a per-beam correlation threshold below which data are to be considered suspect. If the plot type is p, the suspect points (velocity, backscatter amplitude, or correlation) will be coloured red; otherwise, this argument is ignored.
colBrush	Colour to use for brushed (bad) data, if brushCorrelation is active.
main	Main title for plot, used just on the top panel, if there are several panels.
debug	A flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	Optional arguments passed to plotting functions.

Details

Creates a multi-panel summary plot of data measured by an ADV. The panels are controlled by the which argument. (Note the gaps in the sequence, e.g. 4 and 8 are not used.)

- which=1 to 3 (or "u1" to "u3")

  yield timeseries of the first, second, and third components of velocity (in beam, xyz or enu coordinates).
- which=4 is not permitted (since ADV are 3-beam devices)
- which=5 to 7 (or "a1" to "a3") yield timeseries of the amplitudes of beams 1 to 3. (Note that the data are called data$a[,1], data$a[,2] and data$a[,3], for these three timeseries.)
- which=8 is not permitted (since ADV are 3-beam devices)

- `which=9` to `11` (or `"q1"` to `"q3"`) yield timeseries of correlation for beams 1 to 3. (Note that the data are called `data$c[,1]`, `data$c[,2]` and `data$c[,3]`, for these three timeseries.)
- `which=12` is not permitted (since ADVs are 3-beam devices)
- `which=13` is not permitted (since ADVs do not measure salinity)
- `which=14` or `which="temperature"` yields a timeseries of temperature.
- `which=15` or `which="pressure"` yields a timeseries of pressure.
- `which=16` or `which="heading"` yields a timeseries of heading.
- `which=17` or `which="pitch"`yields a timeseries of pitch.
- `which=18` or `which="roll"`yields a timeseries of roll.
- `which=19` to `21` yields plots of correlation versus amplitude, for beams 1 through 3, using `smoothScatter`.
- `which=22` is not permitted (since ADVs are 3-beam devices)
- `which=23` or `"progressive vector"` yields a progressive-vector diagram in the horizontal plane, plotted with `asp=1`, and taking beam1 and beam2 as the eastward and northward components of velocity, respectively.
- `which=28` or `"uv"` yields velocity plot in the horizontal plane, i.e. u[2] versus u[1]. If the number of data points is small, a scattergraph is used, but if it is large, `smoothScatter` is used.
- `which=29` or `"uv+ellipse"` as the `"uv"` case, but with an added indication of the tidal ellipse, calculated from the eigen vectors of the covariance matrix.
- `which=30` or `"uv+ellipse+arrow"` as the `"uv+ellipse"` case, but with an added arrow indicating the mean current.
- `which=50` or `"analog1"` plots a time series of the analog1 signal, if there is one.
- `which=51` or `"analog2"` plots a time series of the analog2 signal, if there is one.
- `which=100` or `"voltage"` plots the voltage as a timeseries, if voltage exists in the dataset.

In addition to the above, there are some groupings defined:

- `which="velocity"` equivalent to `which=1:3` (three velocity components)
- `which="amplitude"` equivalent to `which=5:7` (three amplitude components)
- `which="backscatter"` equivalent to `which=9:11` (three backscatter components)
- `which="hydrography"` equivalent to `which=14:15` (temperature and pressure)
- `which="angles"` equivalent to `which=16:18` (heading, pitch and roll)

Author(s)

Dan Kelley

See Also

The documentation for adv-class explains the structure of ADV objects, and also outlines the other functions dealing with them.

Other functions that plot oce data: plot,adp-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv-class, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

Examples

```
library(oce)
data(adv)
plot(adv)
```

---

plot,amsr-method *Plot an amsr Object*

Description

Plot an amsr Object

Usage

```
## S4 method for signature 'amsr'
plot(x, y, asp, missingColor = list(land = "papayaWhip", none
  = "lightGray", bad = "gray", rain = "plum", ice = "mediumVioletRed"),
  debug = getOption("oceDebug"), ...)
```

Arguments

x	An object inheriting from amsr-class.
y	String indicating the name of the band to plot; if not provided, SST is used; see amsr-class for a list of bands.
asp	Optional aspect ratio for plot.
missingColor	List of colours for problem cases. The names of the elements in this list must be as in the default, but the colours may be changed to any desired values. These default values work reasonably well for SST images, which are the default image, and which employ a blue-white-red blend of colours, no mixture of which matches the default values in missingColor.

debug	A debugging flag, integer.
...	extra arguments passed to imagep, e.g. set col to control colours.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to amsr data: [[<-,amsr-method, amsr-class, composite,amsr-method, download.amsr, read.amsr, subset,amsr-method, summary,amsr-method

Examples

```
## Not run:
d <- read.amsr("f34_20160102v7.2.gz")
asp <- 1/cos(pi*40/180)
plot(d, "SST", col=oceColorsJet, xlim=c(-80,0), ylim=c(20,60), asp=asp)
data(coastlineWorldMedium, package="ocedata")
lines(coastlineWorldMedium[['longitude']], coastlineWorldMedium[['latitude']])

## End(Not run)
```

---

plot,argo-method *Plot Argo Data*

Description

Plot a summary diagram for argo data.

Usage

```
## S4 method for signature 'argo'
plot(x, which = 1, level, coastline = c("best",
  "coastlineWorld", "coastlineWorldMedium", "coastlineWorldFine", "none"),
  cex = 1, pch = 1, type = "p", col, fill = FALSE, projection = NULL,
  mgp = getOption("oceMgp"), mar = c(mgp[1] + 1.5, mgp[1] + 1.5, 1.5, 1.5),
  tformat, debug = getOption("oceDebug"), ...)
```

Arguments

`x`	object inheriting from `argo-class`.
`which`	list of desired plot types, one of the following. Note that `oce.pmatch` is used to try to complete partial character matches, and that an error will occur if the match is not complete (e.g. `"salinity"` matches to both `"salinity ts"` and `"salinity profile"`.). • `which=1` or `which="trajectory"` gives a plot of the argo trajectory, with the coastline, if one is provided. • `which=2` or `"salinity ts"` gives a time series of salinity at the indicated level(s) • `which=3` or `"temperature ts"` gives a time series of temperature at the indicated level(s) • `which=4` or `"TS"` gives a TS diagram at the indicated level(s) • `which=5` or `"salinity profile"` gives a salinity profile of all the data (with S and p trimmed to the 1 and 99 percentiles) • `which=6` or `"temperature profile"` gives a temperature profile (with T and p trimmed to the 1 and 99 percentiles)
`level`	depth pseudo-level to plot, for `which=2` and higher. May be an integer, in which case it refers to an index of depth (1 being the top) or it may be the string "all" which means to plot all data.
`coastline`	character string giving the coastline to be used in an Argo-location map, or `"best"` to pick the one with highest resolution, or `"none"` to avoid drawing the coastline.
`cex`	size of plotting symbols to be used if `type='p'`.
`pch`	type of plotting symbols to be used if `type='p'`.
`type`	plot type, either `"l"` or `"p"`.
`col`	optional list of colours for plotting.
`fill`	Either a logical, indicating whether to fill the land with light-gray, or a colour name. Owing to problems with some projections, the default is not to fill.
`projection`	indication of the projection to be used in trajectory maps. If this is `NULL`, no projection is used, although the plot aspect ratio will be set to yield zero shape distortion at the mean float latitude. If `projection="automatic"`, then one of two projections is used: stereopolar (i.e. `"+proj=stere +lon_0=X"` where `X` is the mean longitude), or Mercator (i.e. `"+proj=merc"`) otherwise. Otherwise, `projection` must be a character string specifying a projection in the notation used by project in the **rgdal**; this will be familiar to many readers as the PROJ.4 notation; see `mapPlot`.
`mgp`	3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
`mar`	value to be used with `par`("mar").
`tformat`	optional argument passed to `oce.plot.ts`, for plot types that call that function. (See `strptime` for the format used.)
`debug`	debugging flag.
`...`	optional arguments passed to plotting functions.

Value

None.

Author(s)

Dan Kelley

References

http://www.argo.ucsd.edu/

See Also

Other things related to argo data: [[,argo-method, [[<-,argo-method, argo-class, argoGrid, argoNames2oceNames, argo, as.argo, handleFlags,argo-method, read.argo, subset,argo-method, summary,argo-method

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Examples

```
library(oce)
data(argo)
tc <- cut(argo[["time"]], "year")
plot(argo, pch=as.integer(tc))
year <- substr(levels(tc), 1, 4)
data(topoWorld)
contour(topoWorld[['longitude']], topoWorld[['latitude']],
        topoWorld[['z']], add=TRUE)
legend("bottomleft", pch=seq_along(year), legend=year, bg="white", cex=3/4)
```

---

plot,bremen-method *Plot a Bremen Object*

Description

Plot a bremen object, i.e. one inheriting from bremen-class. If x seems to be a CTD dataset, uses plot,ctd-method; otherwise, x is assumed to be a lowered-adp object, and a two-panel plot is created with plot,ladp-method to show velocity variation with pressure.

Usage

```
## S4 method for signature 'bremen'
plot(x, type, ...)
```

Arguments

x	A bremen object, e.g. as read by read.bremen.
type	Optional string indicating the type to which x should be coerced before plotting. The choices are ctd and ladp.
...	Other arguments, passed to plotting functions.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to bremen data: [[,bremen-method, [[<-,bremen-method, bremen-class, read.bremen, summary,bremen-method

---

plot,cm-method *Plot CM data*

Description

Creates a multi-panel summary plot of data measured by a current meter.

Usage

```
## S4 method for signature 'cm'
plot(x, which = c(1:2, 7:9), type = "l",
  drawTimeRange = getOption("oceDrawTimeRange"), drawZeroLine = FALSE,
  mgp = getOption("oceMgp"), mar = c(mgp[1] + 1.5, mgp[1] + 1.5, 1.5, 1.5),
  small = 2000, main = "", tformat, debug = getOption("oceDebug"), ...)
```

Arguments

x	an cm object, e.g. as read by read.cm.
which	list of desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of which.
type	type of plot, as for plot.
drawTimeRange	boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.
drawZeroLine	boolean that indicates whether to draw zero lines on velocities.

mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
small	an integer indicating the size of data set to be considered "small", to be plotted with points or lines using the standard plot function. Data sets with more than small points will be plotted with smoothScatter instead.
main	main title for plot, used just on the top panel, if there are several panels.
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	Optional arguments passed to plotting functions.

Details

The panels are controlled by the which argument, as follows.

- which=1 or which="u" for a time-series graph of eastward velocity, u, as a function of time.
- which=2 or which="v" for a time-series graph of northward velocity, u, as a function of time.
- which=3 or "progressive vector" for progressive-vector plot
- which=4 or "uv" for a plot of v versus u. (Dots are used for small datasets, and smoothScatter for large ones.)
- which=5 or "uv+ellipse" as the "uv" case, but with an added indication of the tidal ellipse, calculated from the eigen vectors of the covariance matrix.
- which=6 or "uv+ellipse+arrow" as the "uv+ellipse" case, but with an added arrow indicating the mean current.
- which=7 or "pressure" for pressure
- which=8 or "salinity" for salinity
- which=9 or "temperature" for temperature
- which=10 or "TS" for a TS diagram
- which=11 or "conductivity" for conductivity
- which=20 or "direction" for the direction of flow

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to cm data: [[,cm-method, [[<-,cm-method, as.cm, cm-class, cm, read.cm, subset,cm-method, summary,cm-method

Examples

```
library(oce)
data(cm)
summary(cm)
plot(cm)
```

---

plot,coastline-method *Plot a Coastline*

Description

This function plots a coastline. An attempt is made to fill the space of the plot, and this is done by limiting either the longitude range or the latitude range, as appropriate, by modifying the eastern or northern limit, as appropriate.

Usage

```
## S4 method for signature 'coastline'
plot(x, xlab = "", ylab = "", showHemi = TRUE, asp,
  clongitude, clatitude, span, lonlabel = NULL, latlabel = NULL,
  sides = NULL, projection = NULL, expand = 1,
  mgp = getOption("oceMgp"), mar = c(mgp[1] + 1, mgp[1] + 1, 1, 1), bg,
  fill, type = "polygon", border = NULL, col = NULL, axes = TRUE,
  cex.axis = par("cex.axis"), add = FALSE, inset = FALSE,
  geographical = 0, longitudelim, latitudelim,
  debug = getOption("oceDebug"), ...)
```

Arguments

x	A coastline object, as read by read.coastline or created by as.coastline, or a list containing items named longitude and latitude.
xlab	label for x axis
ylab	label for y axis
showHemi	logical indicating whether to show the hemisphere in axis tick labels.
asp	Aspect ratio for plot. The default is for plot,coastline-method to set the aspect ratio to give natural latitude-longitude scaling somewhere near the centre latitude on the plot. Often, it makes sense to set asp yourself, e.g. to get correct shapes at 45N, use asp=1/cos(45*pi/180). Note that the land mass is not symmetric about the equator, so to get good world views you should set asp=1

or set `ylim` to be symmetric about zero. Any given value of `asp` is ignored, if `clongitude` and `clatitude` are given (or if the latter two are inferred from `projection`.

`clongitude, clatitude`
: optional center latitude of map, in decimal degrees. If both `clongitude` and `clatitude` are provided, or alternatively if they can be inferred from substrings `+lon_0` and `+lat_0` in `projection`, then any provided value of `asp` is ignored, and instead the plot aspect ratio is computed based on the center latitude. If `clongitude` and `clatitude` are known, then `span` must also be provided, and in this case, it is not permitted to also specify `longitudelim` and `latitudelim`.

`span`
: optional suggested diagonal span of the plot, in kilometers. The plotted span is usually close to the suggestion, although the details depend on the plot aspect ratio and other factors, so some adjustment may be required to fine-tune a plot. A value for `span` must be supplied, if `clongitude` and `clatitude` are supplied (or inferred from `projection`).

`lonlabel, latlabel, sides`
: optional vectors of longitude and latitude to label on the indicated sides of plot, passed to `plot,coastline-method`. Using these arguments permits reasonably simple customization. If they are are not provided, reasonable defaults will be used.

`projection`
: optional map projection to use (see the `mapPlot` argument of the same name). If set to `FALSE` then no projection is used, and the data are plotted in a cartesion frame, with aspect ratio set to reduce distortion near the middle of the plot. This option is useful if the coastline produces spurious horizontal lines owing to islands crossing the plot edges (a problem that plagues map projections). If `projection` is not set, a Mercator projection is used for latitudes below about 70 degrees, as if `projection="+proj=merc"` had been supplied, or a Stereopolar one is used as if `projection="+proj=stere"`. Otherwise, `projection` must be a character string identifying a projection accepted by `mapPlot`.

`expand`
: numerical factor for the expansion of plot limits, showing area outside the plot, e.g. if showing a ship track as a coastline, and then an actual coastline to show the ocean boundary. The value of `expand` is ignored if either `xlim` or `ylim` is given.

`mgp`
: 3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.

`mar`
: value to be used with `par("mar")`.

`bg`
: optional colour to be used for the background of the map. This comes in handy for drawing insets (see "details").

`fill`
: a legacy parameter that will be permitted only temporarily; see "History".

`type`
: indication of type; may be `"polygon"`, for a filled polygon, `"p"` for points, `"l"` for line segments, or `"o"` for points overlain with line segments.

`border`
: colour of coastlines and international borders (ignored unless `type="polygon"`.

`col`
: either the colour for filling polygons (if `type="polygon"`) or the colour of the points and line segments (if `type="p"`, `type="l"`, or `type="o"`).

axes	boolean, set to TRUE to plot axes.
cex.axis	value for axis font size factor.
add	boolean, set to TRUE to draw the coastline on an existing plot. Note that this retains the aspect ratio of that existing plot, so it is important to set that correctly, e.g. with asp=1/cos(lat * pi / 180), where clat is the central latitude of the plot.
inset	set to TRUE for use within plotInset. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by plotInset.
geographical	flag indicating the style of axes. If geographical=0, the axes are conventional, with decimal degrees as the unit, and negative signs indicating the southern and western hemispheres. If geographical=1, the signs are dropped, with axis values being in decreasing order within the southern and western hemispheres. If geographical=2, the signs are dropped and the axes are labelled with degrees, minutes and seconds, as appropriate, and hemispheres are indicated with letters. If geographical=3, things are the same as for geographical=2, but the hemisphere indication is omitted.
longitudelim	this and latitudelim provide a second way to suggest plot ranges. Note that these may not be supplied if clongitude, clatitude and span are given.
latitudelim	see longitudelim.
debug	set to TRUE to get debugging information during processing.
...	optional arguments passed to plotting functions. For example, set yaxp=c(-90,90,4) for a plot extending from pole to pole.

Details

If longitudelim, latitudelim and projection are all given, then these arguments are passed to mapPlot to produce the plot. (The call uses bg for col, and uses col, fill and border directly.) If the results need further customization, users should use mapPlot directly.

If projection is provided without longitudelim or latitudelim, then mapPlot is still called, but longitudelim and latitudelim are computed from clongitude, clatitude and span.

If projection is not provided, much simpler plots are produced. These are Cartesian, with aspect ratio set to minimize shape distortion at the central latitude. Although these are crude, they have the benefit of always working, which cannot be said of true map projections, which can be problematic in various ways owing to difficulties in inverting projection calculations.

To get an inset map inside another map, draw the first map, do par(new=TRUE), and then call plot,coastline-method with a value of mar that moves the inset plot to a desired location on the existing plot, and with bg="white".

Value

None.

History

Until February, 2016, `plot,coastline-method` relied on a now-defunct argument `fill` to control colours; `col` is to be used now, instead. Also, in February, 2016, the arguments named `parameters` and `orientation` were both removed, as they had become nonfunctional about a year previously, in the transition to using the `rgdal` package to carry out map projections.

Author(s)

Dan Kelley

See Also

The documentation for `coastline-class` explains the structure of coastline objects, and also outlines the other functions dealing with them.

Other functions that plot oce data: `plot,adp-method`, `plot,adv-method`, `plot,amsr-method`, `plot,argo-method`, `plot,bremen-method`, `plot,cm-method`, `plot,ctd-method`, `plot,gps-method`, `plot,ladp-method`, `plot,lisst-method`, `plot,lobo-method`, `plot,met-method`, `plot,odf-method`, `plot,rsk-method`, `plot,satellite-method`, `plot,sealevel-method`, `plot,section-method`, `plot,tidem-method`, `plot,topo-method`, `plot,windrose-method`, `plotProfile`, `plotScan`, `plotTS`, `tidem-class`

Other things related to `coastline` data: `[[,coastline-method`, `[[<-,coastline-method`, `as.coastline`, `coastline-class`, `coastlineBest`, `coastlineCut`, `coastlineWorld`, `download.coastline`, `read.coastline.openstre`
`read.coastline.shapefile`, `subset,coastline-method`, `summary,coastline-method`

Examples

```
## Not run:
library(oce)
par(mar=c(2, 2, 1, 1))
data(coastlineWorld)
plot(coastlineWorld)
plot(coastlineWorld, clongitude=-63.6, clatitude=44.6, span=1000)

## Canada in Lambert projection
plot(coastlineWorld, clongitude=-95, clatitude=65, span=5500,
     grid=10, projection='+proj=laea +lon_0=-100 +lat_0=55')

## End(Not run)
```

`plot,ctd-method` *Plot CTD Data*

Description

Plot CTD data, by default in a four-panel display showing (a) profiles of salinity and temperature, (b) profiles of density and the square of buoyancy frequency, (c) a TS diagram and (d) a coastline diagram indicating the station location.

Usage

```
## S4 method for signature 'ctd'
plot(x, which, col = par("fg"), fill, borderCoastline = NA,
  colCoastline = "lightgray", eos = getOption("oceEOS", default = "gsw"),
  ref.lat = NaN, ref.lon = NaN, grid = TRUE, coastline = "best", Slim,
  Clim, Tlim, plim, densitylim, N2lim, Rrholim, dpdtlim, timelim, lonlim,
  latlim, drawIsobaths = FALSE, clongitude, clatitude, span,
  showHemi = TRUE, lonlabel = NULL, latlabel = NULL, sides = NULL,
  projection = NULL, parameters = NULL, orientation = NULL,
  latlon.pch = 20, latlon.cex = 1.5, latlon.col = "red", cex = 1,
  cex.axis = par("cex.axis"), pch = 1, useSmoothScatter = FALSE, df,
  keepNA = FALSE, type = "l", mgp = getOption("oceMgp"), mar = c(mgp[1]
  + 1.5, mgp[1] + 1.5, mgp[1] + 1.5, mgp[1] + 1), inset = FALSE,
  add = FALSE, debug = getOption("oceDebug"), ...)
```

Arguments

x — A ctd object, i.e. one inheriting from ctd-class.

which — List of desired plot types, as given below. If which is not supplied, a default will be used. This default will be c(1,2,3,5) if the CTD is in profiling mode (i.e. if deploymentType in the metadata slot equals "profile", or is missing) or "moored"/"thermosalinograph", the default will be c(30, 3, 31, 5). If it is "towyo", c(30, 31, 32, 3) will be used. Details are as follows.

- which=1 or which="salinity+temperature" gives a combined profile of temperature and salinity
- which=2 or which="density+N2" gives a combined profile of σ_θ and N^2
- which=3 or which="TS" gives a TS plot
- which=4 or which="text" gives a textual summary of some aspects of the data
- which=5 or which="map" gives a map plotted with plot,coastline-method, with a dot for the station location. Notes near the top boundary of the map give the station number, the sampling date, and the name of the chief scientist, if these are known. Note that the longitude will be converted to a value between -180 and 180 before plotting. (See also notes about span.)
- which=5.1 as for which=5, except that the file name is drawn above the map
- which=6 or which="density+dpdt" gives a profile of density and dP/dt, which is useful for evaluating whether the instrument is dropping properly through the water column
- which=7 or which="density+time" gives a profile of density and time
- which=8 or which="index" gives a profile of index number (especially useful for ctdTrim)
- which=9 or which="salinity" gives a salinity profile
- which=10 or which="temperature" gives a temperature profile
- which=11 or which="density" gives a density profile
- which=12 or which="N2" gives an N^2 profile

- `which=13` or `which="spice"` gives a spiciness profile
- `which=14` or `which="tritium"` gives a tritium profile
- `which=15` or `which="Rrho"` gives an Rrho profile
- `which=16` or `which="RrhoSF"` gives an RrhoSF profile
- `which=17` or `which="conductivity"` gives a conductivity profile
- `which=20` or `which="CT"` gives a Conservative Temperature profile
- `which=21` or `which="SA"` gives an Absolute Salinity profile
- `which=30` gives a time series of Salinity
- `which=31` gives a time series of Temperature
- `which=32` gives a time series of pressure
- `which=33` gives a time series of sigmaTheta

`col` Colour of lines or symbols.

`fill` A legacy parameter that will be permitted only temporarily; see "History".

`borderCoastline` Colour of coastlines and international borders, passed to `plot,coastline-method` if a map is included in `which`.

`colCoastline` Fill colour of coastlines and international borders, passed to `plot,coastline-method` if a map is included in `which`. Set to `NULL` to avoid filling.

`eos` String indicating the equation of state to be used, either `"unesco"` or `"gsw"`.

`ref.lat` Latitude of reference point for distance calculation.

`ref.lon` Longitude of reference point for distance calculation.

`grid` Set `TRUE` to get a grid on all plots.

`coastline` A specification of the coastline to be used for `which="map"`. This may be a coastline object, whether built-in or supplied by the user, or a character string. If the later, it may be the name of a built-in coastline (`"coastlineWorld"`, `"coastlineWorldFine"`, or `"coastlineWorldCoarse"`), or `"best"`, to choose a suitable coastline for the locale, or `"none"` to prevent the drawing of a coastline. There is a speed penalty for providing `coastline` as a character string, because it forces `plot,coastline-method` to load it on every call. So, if `plot,coastline-method` is to be called several times for a given coastline, it makes sense to load it in before the first call, and to supply the object as an argument, as opposed to the name of the object.

`Slim` Optional limits of salinity axes.

`Clim` Optional limits of conductivity axes.

`Tlim` Optional limits of temperature axes.

`plim` Optional limits of pressure axes.

`densitylim` Optional limits of density axis, whether that axis be horizontal or vertical.

`N2lim` Optional limits of N^2 axis.

`Rrholim` Optional limits of R_rho axis.

`dpdtlim` Optional limits of dP/dt axis.

`timelim` Optional limits of delta-time axis.

`lonlim` Optional limits of longitude axis of map (ignored if no map plotted) DEPRECATED 2014-01-07.

`latlim` Optional limits of latitude axis of map (ignored if no map plotted) DEPRECATED 2014-01-07.

`drawIsobaths` An indication of whether to draw depth contours on maps, in addition to the coastline. The argument has no effect except for panels in which the value of `which` equals `"map"` or the equivalent numerical code, `5`. If `drawIsobaths` is `FALSE`, then no contours are drawn. If `drawIsobaths` is `TRUE`, then contours are selected automatically, using `pretty(c(0,300))` if the station depth is under 100m or `pretty(c(0,5500))` otherwise. If `drawIsobaths` is a numerical vector, then the indicated depths are drawn. For plots drawn with `projection` set to `NULL`, the contours are added with `contour` and otherwise `mapContour` is used. To customize the resultant contours, e.g. setting particular line types or colours, users should call these functions directly (see e.g. Example 2).

`clongitude` Center longitude.

`clatitude` Center latitude.

`span` Optional span of map, in km. If not given, this will be determined as a small multiple of the distance to the nearest point of land, in an attempt to show some coastline in the plot.

`showHemi` Logical indicating whether to show hemisphere in axis tick labels.

`lonlabel, latlabel, sides`
Optional vectors of longitude and latitude to label on the indicated sides of plot, passed to `plot,coastline-method`. Using these arguments permits reasonably simple customization. If they are are not provided, reasonable defaults will be used.

`projection` Projection for map, if desired. If this is `NULL`, no projection will be used; the map will simply show longitude and latitude in a cartesian frame, scaled to retain shapes at the centre. If this is the string `"automatic"`, then either a Mercator or Stereographic projection will be used, depending on whether the CTD station is within 70 degrees of the equator or at higher latitudes. Finally, if this is a string in the format used by `mapPlot`, then it is is passed to that function.

`parameters` a **deprecated** argument that has been ignored since February 2016; see oce-deprecated.

`orientation` a **deprecated** argument that has been ignored since February 2016; see oce-deprecated.

`latlon.pch` Symbol code for sample location (ignored if no map plotted).

`latlon.cex` Symbol expansion factor for sample location (ignored if no map plotted).

`latlon.col` Colour of symbol for sample location (ignored if no map plotted).

`cex` Size to be used for plot symbols (see `par`).

`cex.axis` Size factor for axis labels (see `par`).

`pch` Code for plotting symbol (see `par`).

`useSmoothScatter`
Boolean, set to `TRUE` to use `smoothScatter` instead of `plot` to draw the plot.

- `df` — Optional argument that is ignored except for plotting buoyancy frequency; in that case, it is passed to swN2 as the argument named `df`.
- `keepNA` — Flag indicating whether to keep `NA` values in linegraphs, which will yield breaks in the lines.
- `type` — The type of plot to draw, using the same scheme as `plot`.
- `mgp` — Three-element numerical vector specifying axis-label geometry, passed to `par`. The default establishes tighter margins than in the usual R setup.
- `mar` — Four-element numerical vector specifying margin geometry, passed to `par`. The default establishes tighter margins than in the usual R setup.
- `inset` — Set to `TRUE` for use within `plotInset`. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by `plotInset`.
- `add` — Logical, indication of whether to add to an existing plot. This only works if `length(which)=1`, and it will yield odd results if the value of `which` does not match that in the previous plots.
- `debug` — an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.
- `...` — Optional arguments passed to plotting functions. A common example is to set `df`, for use in swN2 calculations.

Details

Creates a multi-panel summary plot of data measured in a CTD cast. The default values of `which` and other arguments are chosen to be useful for quick overviews of data. However, for detailed work it is common to call the present function with just a single value of `which`, e.g. with four calls to get four panels. The advantage of this is that it provides much more control over the display, and also it permits the addition of extra display elements (lines, points, margin notes, etc.) to the individual panels.

Note that panels that draw more than one curve (e.g. `which="salinity+temperature"` draws temperature and salinity profiles in one graph), the value of `par("usr")` is established by the second profile to have been drawn. Some experimentation will reveal what this profile is, for each permitted `which` case, although it seems unlikely that this will help much ... the simple fact is that drawing two profiles in one graph is useful for a quick overview, but not useful for e.g. interactive analysis with `locator` to flag bad data, etc.

History

Until February, 2016, `plot,ctd-method` relied on a now-defunct argument `fill` to control colours; `colCoastline` is to be used now, instead. Also, now it is possible to set the colour of coasts and international boundaries, with `borderCoastline`.

Author(s)

Dan Kelley

See Also

The documentation for `ctd-class` explains the structure of CTD objects, and also outlines the other functions dealing with them.

Other functions that plot oce data: `plot,adp-method`, `plot,adv-method`, `plot,amsr-method`, `plot,argo-method`, `plot,bremen-method`, `plot,cm-method`, `plot,coastline-method`, `plot,gps-method`, `plot,ladp-method`, `plot,lisst-method`, `plot,lobo-method`, `plot,met-method`, `plot,odf-method`, `plot,rsk-method`, `plot,satellite-method`, `plot,sealevel-method`, `plot,section-method`, `plot,tidem-method`, `plot,topo-method`, `plot,windrose-method`, `plotProfile`, `plotScan`, `plotTS`, `tidem-class`

Other things related to `ctd` data: `[[,ctd-method`, `[[<-,ctd-method`, `as.ctd`, `cnvName2oceName`, `ctd-class`, `ctdDecimate`, `ctdFindProfiles`, `ctdRaw`, `ctdTrim`, `ctd`, `handleFlags,ctd-method`, `oceNames2whpNames`, `oceUnits2whpUnits`, `plotProfile`, `plotScan`, `plotTS`, `read.ctd.itp`, `read.ctd.odf`, `read.ctd.sbe`, `read.ctd.woce.other`, `read.ctd.woce`, `read.ctd`, `subset,ctd-method`, `summary,ctd-method`, `woceNames2oceNames`, `woceUnit2oceUnit`, `write.ctd`

Examples

```
## 1. simple plot
library(oce)
data(ctd)
plot(ctd)

## 2. how to customize depth contours
par(mfrow=c(1,2))
data(section)
stn <- section[["station", 105]]
plot(stn, which='map', drawIsobaths=TRUE)
plot(stn, which='map')
data(topoWorld)
tlon <- topoWorld[["longitude"]]
tlat <- topoWorld[["latitude"]]
tdep <- -topoWorld[["z"]]
contour(tlon, tlat, tdep, drawlabels=FALSE,
        levels=seq(1000,6000,1000), col='lightblue', add=TRUE)
contour(tlon, tlat, tdep, vfont=c("sans serif", "bold"),
        levels=stn[['waterDepth']], col='red', lwd=2, add=TRUE)
```

`plot,echosounder-method` *Plot Echosounder Data*

Description

Plot echosounder data. Simple linear approximation is used when a newx value is specified with the which=2 method, but arguably a gridding method should be used, and this may be added in the future.

Usage

```
## S4 method for signature 'echosounder'
plot(x, which = 1, beam = "a", newx, xlab, ylab,
  xlim, ylim, zlim, type = "l", col = oce.colorsJet, lwd = 2,
  despike = FALSE, drawBottom, ignore = 5, drawTimeRange = FALSE,
  drawPalette = TRUE, radius, coastline, mgp = getOption("oceMgp"),
  mar = c(mgp[1] + 1, mgp[1] + 1, mgp[1] + 1, mgp[1] + 1), atTop, labelsTop,
  tformat, debug = getOption("oceDebug"), ...)
```

Arguments

x	An echosounder object, e.g. as read by read.echosounder, or created by as.echosounder.
which	list of desired plot types: which=1 or which="zt image" gives a z-time image, which=2 or which="zx image" gives a z-distance image, and which=3 or which="map" gives a map showing the cruise track. In the image plots, the display is of log10 of amplitude, trimmed to zero for any amplitude values less than 1 (including missing values, which equal 0). Add 10 to the numeric codes to get the secondary data (non-existent for single-beam files,
beam	a more detailed specification of the data to be plotted. For single-beam data, this may only be "a". For dual-beam data, this may be "a" for the narrow-beam signal, or "b" for the wide-beam signal. For split-beam data, this may be "a" for amplitude, "b" for x-angle data, or "c" for y-angle data.
newx	optional vector of values to appear on the horizontal axis if which=1, instead of time. This must be of the same length as the time vector, because the image is remapped from time to newx using approx.
xlab, ylab	optional labels for the horizontal and vertical axes; if not provided, the labels depend on the value of which.
xlim	optional range for x axis.
ylim	optional range for y axis.
zlim	optional range for colour scale.
type	type of graph, "l" for line, "p" for points, or "b" for both.
col	colour scale for image, a function
lwd	line width (ignored if type="p")
despike	remove vertical banding by using smooth to smooth across image columns, row by row.
drawBottom	optional flag used for section images. If TRUE, then the bottom is inferred as a smoothed version of the ridge of highest image value, and data below that are grayed out after the image is drawn. If drawBottom is a colour, then that colour

	is used, instead of white. The bottom is detected with findBottom, using the ignore value described next.
ignore	optional flag specifying the thickness in metres of a surface region to be ignored during the bottom-detection process. This is ignored unless drawBottom=TRUE.
drawTimeRange	if TRUE, the time range will be drawn at the top. Ignored except for which=2, i.e. distance-depth plots.
drawPalette	if TRUE, the palette will be drawn.
radius	radius to use for maps; ignored unless which=3 or which="map".
coastline	coastline to use for maps; ignored unless which=3 or which="map".
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
atTop	optional vector of time values, for labels at the top of the plot produced with which=2. If labelsTop is provided, then it will hold the labels. If labelsTop is not provided, the labels will be constructed with the format function, and these may be customized by supplying a format in the ... arguments.
labelsTop	optional vector of character strings to be plotted above the atTop times. Ignored unless atTop was provided.
tformat	optional argument passed to imagep, for plot types that call that function. (See strptime for the format used.)
debug	set to an integer exceeding zero, to get debugging information during processing.
...	optional arguments passed to plotting functions. For example, for maps, it is possible to specify the radius of the view in kilometres, with radius.

Value

A list is silently returned, containing xat and yat, values that can be used by oce.grid to add a grid to the plot.

Author(s)

Dan Kelley, with extensive help from Clark Richards

See Also

Other things related to echosounder data: [[,echosounder-method, [[<-,echosounder-method, as.echosounder, echosounder-class, echosounder, findBottom, read.echosounder, subset,echosounder-method, summary,echosounder-method

Examples

```
## Not run:
library(oce)
data(echosounder)
```

```
plot(echosounder, which=c(1,2), drawBottom=TRUE)

## End(Not run)
```

`plot,gps-method` *Plot a GPS Object*

Description

This function plots a gps object. An attempt is made to use the whole space of the plot, and this is done by limiting either the longitude range or the latitude range, as appropriate, by modifying the eastern or northern limit, as appropriate. To get an inset map inside another map, draw the first map, do `par(new=TRUE)`, and then call `plot.gps` with a value of `mar` that moves the inset plot to a desired location on the existing plot, and with `bg="white"`.

Usage

```
## S4 method for signature 'gps'
plot(x, xlab = "", ylab = "", asp, clongitude, clatitude,
  span, projection, parameters = NULL, orientation = NULL, expand = 1,
  mgp = getOption("oceMgp"), mar = c(mgp[1] + 1, mgp[1] + 1, 1, 1), bg,
  axes = TRUE, cex.axis = par("cex.axis"), add = FALSE, inset = FALSE,
  geographical = 0, debug = getOption("oceDebug"), ...)
```

Arguments

`x`	A gps object, as read by `read.gps` or created by `as.gps`, or a list containing items named `longitude` and `latitude`.
`xlab`	label for x axis
`ylab`	label for y axis
`asp`	Aspect ratio for plot. The default is for `plot.gps` to set the aspect ratio to give natural latitude-longitude scaling somewhere near the centre latitude on the plot. Often, it makes sense to set `asp` yourself, e.g. to get correct shapes at 45N, use `asp=1/cos(45*pi/180)`. Note that the land mass is not symmetric about the equator, so to get good world views you should set `asp=1` or set `ylim` to be symmetric about zero. Any given value of `asp` is ignored, if `clongitude` and `clatitude` are given.
`clongitude, clatitude`	optional center latitude of map, in decimal degrees. If both `clongitude` and `clatitude` are provided, then any provided value of `asp` is ignored, and instead the plot aspect ratio is computed based on the center latitude. If `clongitude` and `clatitude` are provided, then `span` must also be provided.
`span`	optional suggested span of plot, in kilometers. The suggestion is an upper limit on the scale; depending on the aspect ratio of the plotting device, the radius may be smaller than `span`. A value for `span` must be supplied, if `clongitude` and `clatitude` are supplied.

projection	optional map projection to use (see mapPlot); if not given, a cartesian frame is used, scaled so that gps shapes near the centre of the plot are preserved. If a projection is provided, the coordinate system will bear an indirect relationship to longitude and longitude, and further adornment of the plot must be done with e.g. mapPoints instead of points.
parameters	optional parameters to map projection (see mapPlot.
orientation	optional orientation of map projection (see mapPlot.
expand	numerical factor for the expansion of plot limits, showing area outside the plot, e.g. if showing a ship track as a gps, and then an actual gps to show the ocean boundary. The value of `expand` is ignored if either `xlim` or `ylim` is given.
mgp	3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with `par("mar")`.
bg	optional colour to be used for the background of the map. This comes in handy for drawing insets (see "details").
axes	boolean, set to `TRUE` to plot axes.
cex.axis	value for axis font size factor.
add	boolean, set to `TRUE` to draw the gps on an existing plot. Note that this retains the aspect ratio of that existing plot, so it is important to set that correctly, e.g. with `asp=1/cos(lat * pi / 180)`, where `clat` is the central latitude of the plot.
inset	set to `TRUE` for use within plotInset. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by plotInset.
geographical	flag indicating the style of axes. If `geographical=0`, the axes are conventional, with decimal degrees as the unit, and negative signs indicating the southern and western hemispheres. If `geographical=1`, the signs are dropped, with axis values being in decreasing order within the southern and western hemispheres. If `geographical=2`, the signs are dropped and the axes are labelled with degrees, minutes and seconds, as appropriate.
debug	set to `TRUE` to get debugging information during processing.
...	optional arguments passed to plotting functions. For example, set `yaxp=c(-90,90,4)` for a plot extending from pole to pole.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to gps data: [[,gps-method, [[<-,gps-method, as.gps, gps-class, read.gps, summary,gps-method

---

plot,ladp-method *Plot an ladp object*

Description

Uses plotProfile to create panels of depth variation of easterly and northerly velocity components.

Usage

```
## S4 method for signature 'ladp'
plot(x, which = c("u", "v"), ...)
```

Arguments

x	A ladp object, e.g. as read by as.ladp.
which	Vector of names of items to be plotted.
...	Other arguments, passed to plotting functions.

Author(s)

Dan Kelley

See Also

Other things related to ladp data: [[,ladp-method, as.ladp, ladp-class, summary,ladp-method

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class `plot,landsat-method` *Plot a landsat Object*

Description

Plot the data within a landsat image, or information computed from the data. The second category includes possibilities such as an estimate of surface temperature and the "terralook" estimate of a natural-colour view.

Usage

```
## S4 method for signature 'landsat'
plot(x, band, which = 1, decimate = TRUE, zlim,
  utm = FALSE, col = oce.colorsPalette, drawPalette = TRUE,
  showBandName = TRUE, alpha.f = 1, red.f = 1.7, green.f = 1.5,
  blue.f = 6, offset = c(0, -0.05, -0.2, 0), transform = diag(c(red.f,
  green.f, blue.f, alpha.f)), debug = getOption("oceDebug"), ...)
```

Arguments

x	A landsat object, e.g. as read by read.landsat.
band	If given, the name of the band. For Landsat-8 data, this may be one of: "aerosol", "blue", "green", "red", "nir", "swir1", "swir2", "panchromatic", "cirrus", "tirs1", or "tirs2". For Landsat-7 data, this may be one of "blue", "green", "red", "nir", "swir1", "tirs1", "tirs2", "swir2", or "panchromatic". For Landsat data prior to Landsat-7, this may be one of "blue", "green", "red", "nir", "swir1", "tirs1", "tirs2", or "swir2". If band is not given, the ("tirs1") will be used if it exists in the object data, or otherwise the first band will be used. In addition to the above, using band="temperature" will plot an estimate of at-satellite brightness temperature, computed from the tirs1 band, and band="terralook" will plot a sort of natural colour by combining the red, green, blue and nir bands according to the formula provided at https://lta.cr.usgs.gov/terralook/what_is_terralook (a website that worked once, but failed as of Feb 2, 2017).
which	Desired plot type; 1=image, 2=histogram.
decimate	An indication of the desired decimation, passed to imagep for image plots. The default yields faster plotting. Some decimation is sensible for full-size images, since no graphical displays can show 16 thousand pixels on a side.
zlim	Either a pair of numbers giving the limits for the colourscale, or "histogram" to have a flattened histogram (i.e. to maximally increase contrast throughout the domain.) If not given, the 1 and 99 percent quantiles are calculated and used as limits.
utm	A logical value indicating whether to use UTS (easting and northing) instead of longitude and latitude on plot.

`col`	Either a function yielding colours, taking a single integer argument with the desired number of colours, or the string `"natural"`, which combines the information in the `red`, `green` and `blue` bands and produces a natural-hue image. In the latter case, the band designation is ignored, and the object must contain the three colour bands.
`drawPalette`	Indication of the type of palette to draw, if any. See `imagep` for details.
`showBandName`	A logical indicating whether the band name is to plotted in the top margin, near the right-hand side.
`alpha.f`	Argument used if `col="natural"`, to adjust colours with `adjustcolor`.
`red.f`	Argument used if `col="natural"`, to adjust colours with `adjustcolor`. Higher values of `red.f` cause red hues to be emphasized (e.g. dry land).
`green.f`	Argument used if `col="natural"`, to adjust colours with `adjustcolor`. Higher values of `green.f` emphasize green hues (e.g. forests).
`blue.f`	Argument used if `band="terralook"`, to adjust colours with `adjustcolor`. Higher values of `blue.f` emphasize blue hues (e.g. ocean).
`offset`	Argument used if `band="terralook"`, to adjust colours with `adjustcolor`.
`transform`	Argument used if `band="terralook"`, to adjust colours with `adjustcolor`.
`debug`	Set to a positive value to get debugging information during processing.
`...`	optional arguments passed to plotting functions.

Details

For Landsat-8 data, the `band` may be one of: `"aerosol"`, `"blue"`, `"green"`, `"red"`, `"nir"`, `"swir1"`, `"swir2"`, `"panchromatic"`, `"cirrus"`, `"tirs1"`, or `"tirs2"`.

For Landsat-7 data, `band` may be one of `"blue"`, `"green"`, `"red"`, `"nir"`, `"swir1"`, `"tirs1"`, `"tirs2"`, `"swir2"`, or `"panchromatic"`.

For Landsat data prior to Landsat-7, `band` may be one of `"blue"`, `"green"`, `"red"`, `"nir"`, `"swir1"`, `"tirs1"`, `"tirs2"`, or `"swir2"`.

If `band` is not given, the (`"tirs1"`) will be used if it exists in the object data, or otherwise the first band will be used.

In addition to the above there are also some pseudo-bands that can be plotted, as follows.

- Setting `band="temperature"` will plot an estimate of at-satellite brightness temperature, computed from the `tirs1` band.
- Setting `band="terralook"` will plot a sort of natural colour by combining the `red`, `green`, `blue` and `nir` bands according to the formula provided at `https://lta.cr.usgs.gov/terralook/what_is_terraloo` (a website that worked once, but failed as of Feb 2, 2017), namely that the `red`-band data are provided as the `red` argument of the `rgb` function, while the `green` argument is computed as 2/3 of the `green`-band data plus 1/3 of the `nir`-band data, and the `blue` argument is computed as 2/3 of the `green`-band data minus 1/3 of the `nir`-band data. (This is not a typo: the `blue` band is not used.)

Author(s)

Dan Kelley

See Also

Other things related to landsat data: [[,landsat-method, landsat-class, landsatAdd, landsatTrim, landsat, read.landsat, summary,landsat-method

---

plot,lisst-method *Plot LISST data*

Description

Creates a multi-panel summary plot of data measured by LISST instrument.

Usage

```
## S4 method for signature 'lisst'
plot(x, which = c(16, 37, 38), tformat,
  debug = getOption("oceDebug"), ...)
```

Arguments

x	a lisst object, e.g. as read by read.lisst.
which	list of desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of which.
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
debug	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies.
...	optional arguments passed to plotting functions.

Details

The panels are controlled by the which argument, as follows.

- which=1 to 32, or which="C1" to "C32" for a time-series graph of the named column (a size class).
- which=33 or which="lts" for a time-series plot of laser transmission sensor.
- which=34 or which="voltage" for a time-series plot of instrument voltage.
- which=35 or which="aux" for a time-series plot of the external auxiliary input.
- which=36 or which="lrs" for a time-series plot of the laser reference sensor.
- which=37 or which="pressure" for a time-series plot of pressure.
- which=38 or which="temperature" for a time-series plot of temperature.
- which=41 or which="transmission" for a time-series plot of transmission, in percent.
- which=42 or which="beam" for a time-series plot of beam-C, in 1/metre.

Author(s)

Dan Kelley

See Also

The documentation for lisst-class explains the structure of lisst objects, and also outlines the other functions dealing with them.

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to lisst data: [[,lisst-method, [[<-,lisst-method, as.lisst, lisst-class, read.lisst, summary,lisst-method

Examples

```
library(oce)
data(lisst)
plot(lisst)
```

---

plot,lobo-method *Plot LOBO data*

Description

Plot a summary diagram for lobo data.

Usage

```
## S4 method for signature 'lobo'
plot(x, which = c(1, 2, 3), mgp = getOption("oceMgp"),
  mar = c(mgp[2] + 1, mgp[1] + 1, 1, mgp[1] + 1.25),
  debug = getOption("oceDebug"), ...)
```

Arguments

x	A lobo object, e.g. as read by read.lobo.
which	A vector of numbers or character strings, indicating the quantities to plot. These are stacked in a single column. The possible values for which are as follows: 1 or "temperature" for a time series of temperature; 2 or "salinity" for salinity; 3 or "TS" for a TS diagram (which uses eos="unesco"), 4 or "u" for a timeseries of the u component of velocity; 5 or "v" for a timeseries of the v component of velocity; 6 or "nitrate" for a timeseries of nitrate concentration; 7 or "fluorescence" for a timeseries of fluorescence value.

`mgp`	3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
`mar`	value to be used with `par("mar")`.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.
`...`	optional arguments passed to plotting functions.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: `plot,adp-method`, `plot,adv-method`, `plot,amsr-method`, `plot,argo-method`, `plot,bremen-method`, `plot,cm-method`, `plot,coastline-method`, `plot,ctd-method`, `plot,gps-method`, `plot,ladp-method`, `plot,lisst-method`, `plot,met-method`, `plot,odf-method`, `plot,rsk-method`, `plot,satellite-method`, `plot,sealevel-method`, `plot,section-method`, `plot,tidem-method`, `plot,topo-method`, `plot,windrose-method`, `plotProfile`, `plotScan`, `plotTS`, `tidem-class`

Other things related to `lobo` data: `[[,lobo-method`, `[[<-,lobo-method`, `as.lobo`, `lobo-class`, `lobo`, `subset,lobo-method`, `summary,lobo-method`

`plot,met-method` *Plot met Data*

Description

Creates a multi-panel summary plot of data measured in a meteorological data set. cast. The panels are controlled by the `which` argument.

Usage

```
## S4 method for signature 'met'
plot(x, which = 1:4, mgp, mar, tformat,
  debug = getOption("oceDebug"))
```

Arguments

x — A met object, e.g. as read by read.met, or a list containing items named salinity and temperature.

which — list of desired plot types.

- which=1 gives a time-series plot of temperature
- which=2 gives a time-series plot of pressure
- which=3 gives a time-series plot of the x (eastward) component of velocity
- which=4 gives a time-series plot of the y (northward) component of velocity
- which=5 gives a time-series plot of speed
- which=6 gives a time-series plot of direction (degrees clockwise from north; note that the values returned by met[["direction"]] must be multiplied by 10 to get the direction plotted)

mgp — A 3-element numerical vector used with par("mgp") to control the spacing of axis elements. The default is tighter than the R default.

mar — A 4-element numerical vector used with par("mar") to control the plot margins. The default is tighter than the R default.

tformat — optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)

debug — an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

If more than one panel is drawn, then on exit from plot.met, the value of par will be reset to the value it had before the function call. However, if only one panel is drawn, the adjustments to par made within plot.met are left in place, so that further additions may be made to the plot.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to met data: [[,met-method, [[<-,met-method, as.met, download.met, met-class, met, read.met, subset,met-method, summary,met-method

Examples

```
library(oce)
data(met)
plot(met, which=3:4)

## Wind speed and direction during Hurricane Juan
## Compare with the final figure in a white paper by Chris Fogarty
## (available at http://www.novaweather.net/Hurricane_Juan_files/McNabs_plot.pdf
## downloaded 2017-01-02).
library(oce)
data(met)
t0 <- as.POSIXct("2003-09-29 04:00:00", tz="UTC")
dt <- 12 * 3600
juan <- subset(met, t0 - dt <= time & time <= t0 + dt)
par(mfrow=c(2,1))
plot(juan, which=5)
abline(v=t0)
plot(juan, which=6)
abline(v=t0)
```

`plot,oce-method` *Plot an oce Object*

Description

This creates a `pairs` plot of the elements in the `data` slot, if there are more than 2 elements there, or a simple xy plot if 2 elements, or a histogram if 1 element.

Usage

```
## S4 method for signature 'oce'
plot(x, y, ...)
```

Arguments

`x`	A basic oce object, i.e. one inheriting from `oce-class`, but not from any subclass of that (because these subclasses go to the subclass plot methods, e.g. a `ctd-class` object would go to `plot,ctd-method`.
`y`	Ignored; only present here because S4 object for generic `plot` need to have a second parameter before the `...` parameter.
`...`	Passed to `hist`, `plot`, or to

Examples

```
library(oce)
o <- new("oce")
o <- oceSetData(o, 'x', rnorm(10))
o <- oceSetData(o, 'y', rnorm(10))
o <- oceSetData(o, 'z', rnorm(10))
plot(o)
```

---

plot,odf-method *Plot an ODF Object*

Description

Plot data contained within an ODF object, using oce.plot.ts to create panels of time-series plots for all the columns contained in the odf object (or just those that contain at least one finite value, if blanks is FALSE). If the object's data slot does not contain time, then pairs is used to plot all the elements in the data slot that contain at least one finite value. These actions are both crude and there are no arguments to control the behaviour, but this function is really just a stop-gap measure, since in practical work odf objects are usually cast to other types, and those types tend to have more useful plots.

Usage

```
## S4 method for signature 'odf'
plot(x, blanks = TRUE, debug = getOption("oceDebug"))
```

Arguments

x	A odf object, e.g. one inheriting from odf-class.
blanks	A logical value that indicates whether to include dummy plots for data items that lack any finite values.
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to odf data: ODF2oce, ODFNames2oceNames, [[,odf-method, [[<-,odf-method, odf-class, read.ctd.odf, read.odf, subset,odf-method, summary,odf-method

---

plot,rsk-method *Plot Rsk Data*

Description

Rsk data may be in many forms, and it is not easy to devise a general plotting strategy for all of them. The present function is quite crude, on the assumption that users will understand their own datasets, and that they can devise plots that are best-suited to their applications. Sometimes, the sensible scheme is to coerce the object into another form, e.g. using `plot(as.ctd(rsk))` if the object contains CTD-like data. Other times, users should extract data from the `rsk` object and construct plots themselves. The idea is to use the present function mainly to get an overview, and for that reason, the default plot type (set by `which`) is a set of time-series plots, because the one thing that is definitely known about `rsk` objects is that they contain a `time` vector in their `data` slot.

Usage

```
## S4 method for signature 'rsk'
plot(x, which = "timeseries", tlim, ylim, xlab, ylab, tformat,
  drawTimeRange = getOption("oceDrawTimeRange"),
  abbreviateTimeRange = getOption("oceAbbreviateTimeRange"),
  useSmoothScatter = FALSE, mgp = getOption("oceMgp"), mar = c(mgp[1] +
  1.5, mgp[1] + 1.5, 1.5, 1.5), main = "", debug = getOption("oceDebug"),
  ...)
```

Arguments

x	`rsk` object, typically result of read.rsk.
which	character indicating desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of `which`.
tlim	optional limits for time axis. If not provided, the value will be inferred from the data.
ylim	optional limits for the y axis. If not provided, the value will be inferred from the data. (It is helpful to specify this, if the auto-scaled value will be inappropriate, e.g. if more lines are to be added later). Note that this is ignored, unless

	`length(which) == 1` and `which` corresponds to one of the data fields. If a multipanel plot of a specific subset of the data fields is desired with `ylim` control, it should be done panel by panel (see Examples).
`xlab`	optional label for x axis.
`ylab`	optional label for y axis.
`tformat`	optional argument passed to `oce.plot.ts`, for plot types that call that function. (See `strptime` for the format used.)
`drawTimeRange`	boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.
`abbreviateTimeRange`	boolean that applies to panels with time as the horizontal axis, indicating whether to abbreviate the second time in the time range (e.g. skipping the year, month, day, etc. if it's the same as the start time).
`useSmoothScatter`	a boolean to cause `smoothScatter` to be used for profile plots, instead of `plot`.
`mgp`	3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
`mar`	value to be used with `par("mar")`.
`main`	main title for plot, used just on the top panel, if there are several panels.
`debug`	a flag that turns on debugging, if it exceeds 0.
`...`	optional arguments passed to plotting functions.

Details

Plots produced are time series plots of the data in the object. The default, `which="timeseries"` plots all data fields, and over-rides any other specification. Specific fields can be plotted by naming the field, e.g. `which="temperature"` to plot a time series of just the temperature field.

Author(s)

Dan Kelley and Clark Richards

See Also

The documentation for `rsk-class` explains the structure of `rsk` objects, and also outlines the other functions dealing with them.

Other functions that plot oce data: `plot,adp-method`, `plot,adv-method`, `plot,amsr-method`, `plot,argo-method`, `plot,bremen-method`, `plot,cm-method`, `plot,coastline-method`, `plot,ctd-method`, `plot,gps-method`, `plot,ladp-method`, `plot,lisst-method`, `plot,lobo-method`, `plot,met-method`, `plot,odf-method`, `plot,satellite-method`, `plot,sealevel-method`, `plot,section-method`, `plot,tidem-method`, `plot,topo-method`, `plot,windrose-method`, `plotProfile`, `plotScan`, `plotTS`, `tidem-class`

Other things related to `rsk` data: `[[,rsk-method`, `[[<-,rsk-method`, `as.rsk`, `read.rsk`, `rsk-class`, `rskPatm`, `rskToc`, `rsk`, `subset,rsk-method`, `summary,rsk-method`

Examples

```
library(oce)
data(rsk)
plot(rsk) # default timeseries plot of all data fields

## A multipanel plot of just pressure and temperature with ylim
par(mfrow=c(2, 1))
plot(rsk, which="pressure", ylim=c(10, 30))
plot(rsk, which="temperature", ylim=c(2, 4))
```

---

plot,satellite-method *Plot a satellite object*

Description

For an example using g1sst data, see read.g1sst.

Usage

```
## S4 method for signature 'satellite'
plot(x, y, asp, debug = getOption("oceDebug"), ...)
```

Arguments

x	An object inheriting from satellite-class.
y	String indicating the quantity to be plotted.
asp	Optional aspect ratio for plot.
debug	A debugging flag, integer.
...	extra arguments passed to imagep, e.g. set col to control colours.

Author(s)

Dan Kelley

See Also

Other things related to satellite data: g1sst-class, read.g1sst, satellite-class, summary,satellite-method

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

plot,sealevel-method *Plot Sealevel Data*

Description

Creates a plot for a sea-level dataset, in one of two varieties. Depending on the length of `which`, either a single-panel or multi-panel plot is drawn. If there is just one panel, then the value of `par` used in `plot,sealevel-method` is retained upon exit, making it convenient to add to the plot. For multi-panel plots, `par` is returned to the value it had before the call.

Usage

```
## S4 method for signature 'sealevel'
plot(x, which = 1:3,
  drawTimeRange = getOption("oceDrawTimeRange"), mgp = getOption("oceMgp"),
  mar = c(mgp[1] + 0.5, mgp[1] + 1.5, mgp[2] + 1, mgp[2] + 3/4),
  marginsAsImage = FALSE, debug = getOption("oceDebug"), ...)
```

Arguments

`x`	an object of class `"sealevel"`, e.g. as read by `read.sealevel`.
`which`	a numerical or string vector indicating desired plot types, with possibilities 1 or `"all"` for a time-series of all the data, 2 or `"month"` for a time-series of just the first month, 3 or `"spectrum"` for a power spectrum (truncated to frequencies below 0.1 cycles per hour, or 4 or `"cumulativespectrum"` for a cumulative integral of the power spectrum.
`drawTimeRange`	boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.
`mgp`	3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
`mar`	value to be used with `par("mar")`.
`marginsAsImage`	boolean, `TRUE` to put a wide margin to the right of time-series plots, matching the space used up by a palette in an `imagep` plot.
`debug`	a flag that turns on debugging, if it exceeds 0.
`...`	optional arguments passed to plotting functions.

Value

None.

Author(s)

Dan Kelley

References

The example refers to Hurricane Juan, which caused a great deal of damage to Halifax in 2003. Since this was in the era of the digital photo, a casual web search will uncover some spectacular images of damage, from both wind and storm surge. A map of the path of Hurricane Juan across Nova Scotia is at http://ec.gc.ca/ouragans-hurricanes/default.asp?lang=En&n=222F51F7-1. Landfall, very near the site of this sealevel gauge, was between 00:10 and 00:20 Halifax local time on Monday, Sept 29, 2003.

See Also

The documentation for `sealevel-class` explains the structure of sealevel objects, and also outlines the other functions dealing with them.

Other functions that plot oce data: `plot,adp-method`, `plot,adv-method`, `plot,amsr-method`, `plot,argo-method`, `plot,bremen-method`, `plot,cm-method`, `plot,coastline-method`, `plot,ctd-method`, `plot,gps-method`, `plot,ladp-method`, `plot,lisst-method`, `plot,lobo-method`, `plot,met-method`, `plot,odf-method`, `plot,rsk-method`, `plot,satellite-method`, `plot,section-method`, `plot,tidem-method`, `plot,topo-method`, `plot,windrose-method`, `plotProfile`, `plotScan`, `plotTS`, `tidem-class`

Other things related to sealevel data: `[[,sealevel-method`, `[[<-,sealevel-method`, `as.sealevel`, `read.sealevel`, `sealevel-class`, `sealevelTuktoyaktuk`, `sealevel`, `subset,sealevel-method`, `summary,sealevel-method`

Examples

```
library(oce)
data(sealevel)
## local Halifax time is UTC + 4h; see [2] on timing
juan <- as.POSIXct("2003-09-29 00:15:00", tz="UTC")+4*3600
plot(sealevel, which=1, xlim=juan+86400*c(-7, 7))
abline(v=juan, col='red')
```

`plot,section-method` *Plot a Section*

Description

Creates a summary plot for a CTD section, with one panel for each value of `which`.

Usage

```
## S4 method for signature 'section'
plot(x, which = c(1, 2, 3, 99), eos, at = NULL,
  labels = TRUE, grid = FALSE, contourLevels = NULL,
  contourLabels = NULL, stationIndices, coastline = "best", xlim = NULL,
  ylim = NULL, zlim = NULL, map.xlim = NULL, map.ylim = NULL,
  clongitude, clatitude, span, projection = NULL, xtype = "distance",
  ytype = "depth", ztype = "contour", zbreaks = NULL, zcol = NULL,
```

```
legend.loc = "bottomright", showStations = FALSE, showStart = TRUE,
showBottom = TRUE, axes = TRUE, mgp, mar, col, cex, pch, debug, ...)
```

Arguments

`x` a `section` object, e.g. as created by `as.section` or `read.section`.

`which` a list of desired plot types, as explained in "Details". There may be up to four panels in total, and the desired plots are placed in these panels, in reading order. If only one panel is plotted, `par` is not adjusted, which makes it easy to add to the plot with subsequent plotting commands.

`eos` Character indication of the seawater equation of state to use. The permitted choices are `"gsw"` and `"unesco"`. If `eos` is not supplied, it defaults to `getOption("oceEOS", default="g`

`at` If `NULL` (the default), the x axis will indicate the distance of the stations from the first in the section. (This may give errors in the contouring routine, if the stations are not present in a geographical order.) If a list, then it indicates the values at which stations will be plotted.

`labels` Either a logical, indicating whether to put labels on the x axis, or a vector that is a list of labels to be placed at the x positions indicated by `at`.

`grid` If `TRUE`, points are drawn at data locations.

`contourLevels` Optional contour levels.

`contourLabels` Optional contour labels.

`stationIndices` Optional list of the indices of stations to use. Note that an index is *not* a station number, e.g. to show the first 4 stations, use `station.indices=1:4`.

`coastline` String giving the coastline to be used in a station map The permitted choices are `"best"` (the default) to pick a variant that suits the scale, `"coastlineWorld"` for the coarse version that is provided by **oce**, `"coastlineWorldMedium"` or `"coastlineWorldFine"` for two coastlines provided by the **ocedata** package, or `"none"`, to avoid drawing a coastline.

`xlim` Optional limit for x axis (only in sections, not map).

`ylim` Optional limit for y axis (only in sections, not map)

`zlim` Optional two-element numerical vector specifying the limit on the plotted field. This is used only if `ztype="image"`; see also `zbreaks` and `zcol`.

`map.xlim, map.ylim`
Optional limits for station map; `map.ylim` is ignored if `map.xlim` is provided.

`clongitude, clatitude, span`
Optional map centre position and span (km).

`projection` Parameter specifying map projection; see `mapPlot`. If `projection="automatic"`, however, a projection is devised from the data, with `stereographic` if the mean latitude exceeds 70N and `mollweide` otherwise.

`xtype` Type of x axis, for contour plots, either `"distance"` for distance (in km) to the first point in the section, `"track"` for distance along the cruise track, `"longitude"`, `"latitude"`, or `"time"`. Note that if the x values are not in order, they will be put in order (which may make no sense) and a warning will be printed.

ytype	Type of y axis for contour plots, either "pressure" for pressure (in dbar, with zero at the surface) or "depth" for depth (in m below the surface, calculated from pressure with swDepth).
ztype	String indicating whether to how to indicate the "z" data (in the R sense, i.e. this could be salinity, temperature, etc; it does not mean the vertical coordinate) The choices are: "contour" for contours, "image" for an image (drawn with imagep with filledContours=TRUE), or "points" to draw points. In the first two cases, the data must be gridded, with identical pressures at each station.
zbreaks, zcol	Indication of breaks and colours to be used if ztype="points" or "image". If not provided, reasonable default are used. If zlim is given but breaks is not given, then breaks is computed to run from zlim[1] to zlim[2]. If zcol is a function, it will be invoked with an argument equal to 1+length(zbreaks).
legend.loc	Location of legend, as supplied to legend, or set to the empty string to avoid plotting a legend.
showStations	Logical indicating whether to draw station numbers on maps.
showStart	Logical indicating whether to indicate the first station with a different symbol than the others.
showBottom	An indication of whether (and how) to indicate the ocean bottom. If FALSE, then the bottom is not rendered. If TRUE, then it is rendered with a gray polygon. If showBottom is a character string, then there are three possibilities: is the string is "polygon" then a polygon is drawn, if it is "lines" then a line is drawn, and if it is "points" then points are drawn. If showBottom is an object inheriting from topo-class then the station locations are interpolated to that topography and the results are shown with a polygon. In this last case, the interpolation is set at a grid that is roughly in accordance with the resolution of the latitudes in the topo object. See "Examples".
axes	Logical value indicating whether to draw axes.
mgp	A 3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. If not provided, this defaults to getOption("oceMgp").
mar	Value to be used with par("mar"). If not provided, a default is set up.
col	Colour, which defaults to par("col").
cex	Numerical character-expansion factor, which defaults to par("cex").
pch	Indication of symbol type; defaults to par("pch").
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If debug is not supplied, it defaults to getOption("oceDebug").
...	Optional arguments passed to the contouring function, e.g. using labcex=1 will increase the size of contour labels.

Details

The type of plot is governed by which, as follows.

- which=0 or "potential temperature" for temperature contours
- which=1 or "temperature" for temperature contours (the default)
- which=2 or "salinity" for salinity contours
- which=3 or "sigmaTheta" for sigma-theta contours
- which=4 or "nitrate" for nitrate concentration contours
- which=5 or "nitrite" for nitrite concentration contours
- which=6 or "oxygen" for oxygen concentration contours
- which=7 or "phosphate" for phosphate concentration contours
- which=8 or "silicate" for silicate concentration contours
- which=9 or "u" for eastward velocity
- which=10 or "uz" for vertical derivative of eastward velocity
- which=11 or "v" for northward velocity
- which=12 or "vz" for vertical derivative of northward velocity
- which=20 or "data" for a dot for each data location
- which=99 or "map" for a location map

The y-axis for the contours is pressure, plotted in the conventional reversed form, so that the water surface appears at the top of the plot. The x-axis is more complicated. If at is not supplied, then the routine calculates x as the distance between the first station in the section and each of the other stations. (This will produce an error if the stations are not ordered geographically, because the contour routine cannot handle non-increasing axis coordinates.) If at is specified, then it is taken to be the location, in arbitrary units, along the x-axis of labels specified by labels; the way this works is designed to be the same as for axis.

Value

If the original section was gridded, the return value is that section. Otherwise, the gridded section that was constructed for the plot is returned. In both cases, the value is returned silently. The purpose of returning the section is to enable subsequent processing of the grid, including adding elements to the plot (see example 5).

Author(s)

Dan Kelley

See Also

The documentation for section-class explains the structure of section objects, and also outlines the other functions dealing with them.

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to section data: [[,section-method, [[<-,section-method, as.section, handleFlags,section-method, read.section, section-class, sectionAddStation, sectionGrid, sectionSmooth, sectionSort, section, subset,section-method, summary,section-method

Examples

```
library(oce)
data(section)
sg <- sectionGrid(section)

## 1. AO3 section, default fields.
plot(section)

## 2. Gulf Stream
GS <- subset(section, 109<=stationId&stationId<=129)
GSg <- sectionGrid(GS, p=seq(0, 2000, 100))
plot(GSg, which=c(1, 99), map.ylim=c(34, 42))
par(mfrow=c(2, 1))
plot(GS, which=1, ylim=c(2000, 0), ztype='points',
     zbreaks=seq(0,30,2), pch=20, cex=3)
plot(GSg, which=1, ztype='image', zbreaks=seq(0,30,2))

par(mfrow=c(1, 1))

## 3. Image, with coloured dots to indicate grid-data mismatch.
plot(GSg, which=1, ztype='image')
T <- GS[['temperature']]
col <- oce.colorsJet(100)[rescale(T, rlow=1, rhigh=100)]
points(GS[['distance']],GS[['depth']],pch=20,cex=3,col='white')
points(GS[['distance']],GS[['depth']],pch=20,cex=2.5,col=col)


## Not run:
## 4. Image of Absolute Salinity, with 4-minute bathymetry
## It's easy to calculate the desired area for the bathymetry,
## but for brevity we'll hard-code it. Note that download.topo()
## caches the file locally.
f <- download.topo(west=-80, east=0, south=35, north=40, resolution=4)
t <- read.topo(f)
plot(section, which="SA", xtype="longitude", ztype="image", showBottom=t)

## End(Not run)

## Not run:
## 5. Temperature with salinity added in red
s <- plot(section, which="temperature")
distance <- s[["distance", "byStation"]]
depth <- s[["station", 1]][["depth"]]
salinity <- matrix(s[["salinity"]], byrow=TRUE, nrow=length(s[["station"]]))
contour(distance, depth, salinity, col=2, add=TRUE)

## End(Not run)
```

`plot,tidem-method` *Plot a Tidem Prediction*

Description

Plot a summary diagram for a tidal fit.

Usage

```
## S4 method for signature 'tidem'
plot(x, which = 1, constituents = c("SA", "O1", "K1",
  "M2", "S2", "M4"), sides = NULL, col = "blue", log = "",
  mgp = getOption("oceMgp"), mar = c(mgp[1] + 1, mgp[1] + 1, mgp[2] + 0.25,
  mgp[2] + 1), ...)
```

Arguments

x	A tidem object, i.e. one inheriting from tidem-class.
which	integer flag indicating plot type, 1 for stair-case spectral, 2 for spike spectral.
constituents	a character vector of constituents that are to be drawn and label. If NULL, then no constituents will be shown. Consult the built-in dataset tidedata for the permissible constituent names and their frequencies.
sides	an integer vector of length equal to that of constituents, designating the side on which the constituent labels are to be drawn. As in all R graphics, the value 1 indicates the bottom of the plot, and 3 indicates the top. If sides=NULL, the default, then all labels are drawn at the top. Any value of sides that is not either 1 or 3 is converted to 3.
col	a character vector naming colours to be used for constituents. Ignored if sides=3. Repeated to be of the same length as constituents, otherwise.
log	if set to "x", the frequency axis will be logarithmic.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
...	optional arguments passed to plotting functions.

Historical note

An argument named labelIf was removed in July 2016, because it was discovered never to have worked as documented, and because the more useful argument constituents had been added.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to tidem data: [[,tidem-method, [[<-,tidem-method, predict.tidem, summary,tidem-method, tidedata, tidem-class, tidemAstron, tidemVuf, tidem

Examples

```
## Not run:
library(oce)
data(sealevel)
tide <- tidem(sealevel)
plot(tide)

## End(Not run)
```

---

plot,topo-method *Plot a Topo Object*

Description

This plots contours of topographic elevation. The plot aspect ratio is set based on the middle latitude in the plot. The line properties, such as `land.lwd`, may either be a single item, or a vector; in the latter case, the length must match the length of the corresponding properties, e.g. `land.z`.

Usage

```
## S4 method for signature 'topo'
plot(x, xlab = "", ylab = "", asp, clongitude, clatitude,
  span, expand = 1.5, water.z, col.water, lty.water, lwd.water, land.z,
  col.land, lty.land, lwd.land, geographical = FALSE, location = "topright",
  mgp = getOption("oceMgp"), mar = c(mgp[1] + 1, mgp[1] + 1, 1, 1),
  debug = getOption("oceDebug"), ...)
```

Arguments

x	A topo object, i.e. inheriting from topo-class.
xlab, ylab	Character strings giving a label for the x and y axes.
asp	Aspect ratio for plot. The default is for `plot.coastline` to set the aspect ratio to give natural latitude-longitude scaling somewhere near the centre latitude on the plot. Often, it makes sense to set `asp` yourself, e.g. to get correct shapes at 45N, use `asp=1/cos(45*pi/180)`. Note that the land mass is not symmetric about the equator, so to get good world views you should set `asp=1` or set `ylim` to be symmetric about zero. Any given value of `asp` is ignored, if `clongitude` and `clatitude` are given.
clongitude	Optional center longitude of map, in degrees east; see `clatitude`.

clatitude	Optional center latitude of map, in degrees north. If this and clongitude are provided, then any provided value of asp is ignored, and instead the plot aspect ratio is computed based on the center latitude. Also, if clongitude and clatitude are provided, then span must be, also.
span	Optional suggested span of plot, in kilometers (must be supplied, if clongitude and clatitude are supplied).
expand	Numerical factor for the expansion of plot limits, showing area outside the plot, e.g. if showing a ship track as a coastline, and then an actual coastline to show the ocean boundary. The value of expand is ignored if either xlim or ylim is given.
water.z	Depths at which to plot water contours. If not provided, these are inferred from the data.
col.water	Colours corresponding to water.z values. If not provided, these will be "fill" colours from oce.colorsGebco.
lty.water	Line type(s) for water contours.
lwd.water	Line width(s) for water contours.
land.z	Depths at which to plot land contours. If not provided, these are inferred from the data. If set to NULL, no land contours will be plotted.
col.land	Colours corresponding to land.z values. If not provided, these will be "fill" colours from oce.colorsGebco.
lty.land	Line type(s) for land contours.
lwd.land	Line width(s) for land contours.
geographical	Logical, indicating whether to plot latitudes and longitudes without minus signs.
location	Location for a legend (or "none", for no legend).
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	Four-element numericl vector to be used with par("mar").
debug	Numerical value, with positive values indicating higher levels of debugging.
...	Additional arguments passed on to plotting functions.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,windrose-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to topo data: [[,topo-method, [[<-,topo-method, as.topo, download.topo, read.topo, subset,topo-method, summary,topo-method, topo-class, topoInterpolate, topoWorld

Examples

```
library(oce)
data(topoWorld)
plot(topoWorld, clongitude=-60, clatitude=45, span=10000)
```

---

plot,windrose-method *Plot Windrose data*

Description

Plot a windrose object, i.e. one inheriting from windrose-class.

Usage

```
## S4 method for signature 'windrose'
plot(x, type = c("count", "mean", "median", "fivenum"),
  convention = c("meteorological", "oceanographic"),
  mgp = getOption("oceMgp"), mar = c(mgp[1], mgp[1], 1 + mgp[1], mgp[1]),
  col, ...)
```

Arguments

x	A windrose object, e.g. inheriting from windrose-class.
type	The thing to be plotted, either the number of counts in the angle interval, the mean of the values in the interval, the median of the values, or a fivenum representation of the values.
convention	String indicating whether to use meteorological convention or oceanographic convention for the arrows that emanate from the centre of the rose. In meteorological convection, an arrow emanates towards the right on the diagram if the wind is from the east; in oceanographic convention, such an arrow indicates flow *to* the east.
mgp	Three-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	Four-element numerical vector to be used with par("mar").
col	Optional list of colours to use. If not set, the colours will be c("red", "pink", "blue", "lightgray"). For the first three types of plot, the first colour in this list is used to fill in the rose, the third is used for the petals of the rose, and the fourth is used for grid lines. For the "fivenum" type, the first colour is used for the inter-quartile range, the second is used outside this range, the third is used for the median, and the fourth is, again, used for the grid lines.
...	Optional arguments passed to plotting functions.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plotProfile, plotScan, plotTS, tidem-class

Other things related to windrose data: [[,windrose-method, [[<-,windrose-method, as.windrose, summary,windrose-method, windrose-class

Examples

```
library(oce)
opar <- par(no.readonly = TRUE)
xcomp <- rnorm(360) + 1
ycomp <- rnorm(360)
wr <- as.windrose(xcomp, ycomp)
par(mfrow=c(1, 2))
plot(wr)
plot(wr, "fivenum")
par(opar)
```

---

plotInset *Plot an Inset Diagram*

Description

Adds an inset diagram to an existing plot. Note that if the inset is a map or coastline, it will be necessary to supply `inset=TRUE` to prevent the inset diagram from occupying the whole device width. After `plotInset()` has been called, any further plotting will take place within the inset, so it is essential to finish a plot before drawing an inset.

Usage

```
plotInset(xleft, ybottom, xright, ytop, expr, mar = c(2, 2, 1, 1),
  debug = getOption("oceDebug"))
```

Arguments

xleft	location of left-hand of the inset diagram, in the existing plot units. (PROVISIONAL FEATURE: this may also be "bottomleft", to put the inset there. Eventually, other positions may be added.)
ybottom	location of bottom side of the inset diagram, in the existing plot units.

xright	location of right-hand side of the inset diagram, in the existing plot units.
ytop	location of top side of the inset diagram, in the existing plot units.
expr	An expression that draws the inset plot. This may be a single plot command, or a sequence of commands enclosed in curly braces.
mar	margins, in line heights, to be used at the four sides of the inset diagram. (This is often helpful to save space.)
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Author(s)

Dan Kelley

Examples

```
library(oce)
## power law in linear and log form
x <- 1:10
y <- x^2
plot(x, y, log='xy',type='l')
plotInset(3, 1, 10, 8,
          expr=plot(x,y,type='l',cex.axis=3/4,mgp=c(3/2, 1/2, 0)),
          mar=c(2.5, 2.5, 1, 1))

## CTD data with location
data(ctd)
plot(ctd, which="TS")
plotInset(29.9, 2.7, 31, 10,
          expr=plot(ctd, which='map',
          coastline="coastlineWorld",
          span=5000, mar=NULL, cex.axis=3/4))
```

---

plotPolar *Draw a Polar Plot*

Description

Creates a crude polar plot.

Usage

```
plotPolar(r, theta, debug = getOption("oceDebug"), ...)
```

Arguments

`r`	radii of points to plot.
`theta`	angles of points to plot, in degrees.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	optional arguments passed to the lower-level plotting functions.

Author(s)

Dan Kelley

Examples

```
library(oce)
r <- rnorm(50, mean=2, sd=0.1)
theta <- runif(50, 0, 360)
plotPolar(r, theta)
```

`plotProfile` *Plot a CTD Profile*

Description

Plot a profile, showing variation of some quantity (or quantities) with pressure, using the oceanographic convention of putting lower pressures nearer the top of the plot. This works for any `oce` object that has a pressure column in its `data` slot. The colours (`col.salinity`, etc.) are only used if two profiles appear on a plot.

Usage

```
plotProfile(x, xtype = "salinity+temperature", ytype = "pressure",
  eos = getOption("oceEOS", default = "gsw"), lty = 1, xlab = NULL,
  ylab = NULL, col = "black", col.salinity = "darkgreen",
  col.temperature = "red", col.rho = "blue", col.N2 = "brown",
  col.dpdt = "darkgreen", col.time = "darkgreen", pt.bg = "transparent",
  grid = TRUE, col.grid = "lightgray", lty.grid = "dotted", Slim, Clim,
  Tlim, densitylim, N2lim, Rrholim, dpdtlim, timelim, plim, xlim, ylim,
  lwd = par("lwd"), xaxs = "r", yaxs = "r", cex = 1, pch = 1,
  useSmoothScatter = FALSE, df, keepNA = FALSE, type = "l",
  mgp = getOption("oceMgp"), mar, add = FALSE, inset = FALSE,
  debug = getOption("oceDebug", 0), ...)
```

Arguments

- `x` — A `ctd` object, i.e. one inheriting from `ctd-class`.
- `xtype` — Item(s) plotted on the x axis, either a vector of length equal to that of `pressure` in the `data` slot, or a text code from the list below.
  - `"salinity"` Profile of salinity.
  - `"conductivity"` Profile of conductivity.
  - `"temperature"` Profile of *in-situ* temperature.
  - `"theta"` Profile of potential temperature.
  - `"density"` Profile of density (expressed as σ_θ).
  - `"index"` Index of sample (useful for working with `ctdTrim`).
  - `"salinity+temperature"` Profile of salinity and temperature within a single axis frame.
  - `"N2"` Profile of square of buoyancy frequency N^2, calculated with `swN2` with an optional argument setting of `df=length(x[["pressure"]])/4` to do some smoothing.
  - `"density+N2"` Profile of sigma0 and the square of buoyancy frequency within a single axis frame.
  - `"density+dpdt"` Profile of sigma0 and dP/dt for the sensor. The latter is useful in indicating problems with the deployment. It is calculated by first differencing pressure and then using a smoothing spline on the result (to avoid grid-point wiggles that result because the SBE software only writes 3 decimal places in pressure). Note that dP/dt may be off by a scale factor; this should not be a problem if there is a `time` column in the `data` slot, or a `sample.rate` in the `metadata` slot.
  - `"sigma0"`, `"sigma1"`, `"sigma2"`, `"sigma3"` and `"sigma4"` Profile of potential density referenced to 0dbar (i.e. the surface), 1000dbar, 2000dbar, 3000dbar, and 4000dbar.
  - `"spice"` Profile of spice.
  - `"Rrho"` Profile of Rrho, defined in the diffusive sense.
  - `"RrhoSF"` Profile of Rrho, defined in the salt-finger sense.
- `ytype` — variable to use on y axis. The valid choices are: `"pressure"` (the default), `"z"`, `"depth"` and `"sigmaTheta"`.
- `eos` — equation of state to be used, either `"unesco"` or `"gsw"`.
- `lty` — line type for the profile.
- `xlab` — optional label for x axis (at top of plot).
- `ylab` — optional label for y axis. Set to `""` to prevent labelling the axis.
- `col` — colour for a general profile.
- `col.salinity` — colour for salinity profile (see "Details").
- `col.temperature` — colour for temperature (see "Details").
- `col.rho` — colour for density (see "Details").
- `col.N2` — colour for square of buoyancy frequency (see "Details").

col.dpdt	colour for dP/dt.
col.time	colour for delta-time.
pt.bg	inside colour for symbols with pch in 21:25
grid	logical, set to TRUE to get a grid.
col.grid	colour for grid.
lty.grid	line type for grid.
Slim	Optional limit for S axis
Clim	Optional limit for conductivity axis
Tlim	Optional limit for T axis
densitylim	Optional limit for density axis
N2lim	Optional limit for N2 axis
Rrholim	Optional limit for Rrho axis
dpdtlim	Optional limit for dp/dt axis
timelim	Optional limit for delta-time axis
plim	Optional limit for pressure axis, ignored unless ytype=="pressure", in which case it takes precedence over ylim.
xlim	Optional limit for x axis, which can apply to any plot type. This is ignored if the plotted x variable is something for which a limit may be specified with an argument, e.g. xlim is ignored for a salinity profile, because Slim ought to be given in such a case.
ylim	Optional limit for y axis, which can apply to any plot type, although is overridden by plim if ytype is "pressure" or by densitylim if ytype is "sigmaTheta".
lwd	lwd value for data line
xaxs	value of par xaxs to use
yaxs	value of par yaxs to use
cex	size to be used for plot symbols (see par)
pch	code for plotting symbol (see par).
useSmoothScatter	boolean, set to TRUE to use smoothScatter instead of plot to draw the plot.
df	optional argument, passed to swN2 if provided, and if a plot using N^2 is requested.
keepNA	FALSE
type	type of plot to draw, using the same scheme as plot.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	Four-element numerical value to be used to set the plot margins, with a call to par(mar) prior to the plot. If this is not supplied, a reasonable default will be set up.

add	A logical value that controls whether to add to an existing plot. (It makes sense to use add=TRUE in the panel argument of a coplot, for example.)
inset	A logical value indicating whether to draw an inset plot. Setting this to TRUE will prevent the present function from adjusting the margins, which is necessary because margin adjustment is the basis for the method used by plotInset.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to other functions. A common example is to set df, for use in swN2 calculations.

Value

None.

Author(s)

Dan Kelley

See Also

read.ctd scans ctd information from a file, plot,ctd-method is a general plotting function for ctd objects, and plotTS plots a temperature-salinity diagrams.

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotScan, plotTS, tidem-class

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
library(oce)
data(ctd)
plotProfile(ctd, xtype="temperature")
```

plotScan *Plot CTD data in a Low-Level Fashion*

Description

Plot CTD data as time-series against scan number, to help with trimming extraneous data from a CTD cast.

Usage

```
plotScan(x, which = 1, xtype = "scan", type = "l",
  mgp = getOption("oceMgp"), mar = c(mgp[1] + 1.5, mgp[1] + 1.5, mgp[1],
  mgp[1]), ...)
```

Arguments

x	A ctd object, i.e. one inheriting from ctd-class.
which	Numerical vector numerical codes specifying the panels to draw: 1 for pressure vs scan, 2 for diff(pressure) vs scan, 3 for temperature vs scan, and 4 for salinity vs scan.
xtype	Character string indicating variable for the x axis. May be "scan" (the default) or "time". In the former case, a scan variable will be created using seq_along, if necessary. In the latter case, an error results if the data slot of x lacks a variable called time.
type	Character indicating the line type, as for plot.default. The default is "l", meaning to connect data with line segments. Another good choice is "o", to add points at the data.
mgp	Three-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	Four-element vector be used with par("mar"). If set to NULL, then par("mar") is used. A good choice for a TS diagram with a palette to the right is mar=par("mar")+c(0, 0, 0, 1)).
...	Optional arguments passed to plotting functions.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotTS, tidem-class

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
library(oce)
data(ctdRaw)
plotScan(ctdRaw)
abline(v=c(130, 350), col='red') # useful for ctdTrim()
```

---

plotSticks *Draw a Stick Plot*

Description

The arrows are drawn with directions on the graph that match the directions indicated by the u and v components. The arrow size is set relative to the units of the y axis, according to the value of yscale, which has the unit of v divided by the unit of y. The interpretation of diagrams produced by plotSticks can be difficult, owing to overlap in the arrows. For this reason, it is often a good idea to smooth u and v before using this function.

Usage

```
plotSticks(x, y, u, v, yscale = 1, add = FALSE, length = 1/20,
  mgp = getOption("oceMgp"), mar = c(mgp[1] + 1, mgp[1] + 1, 1, 1 +
  par("cex")), xlab = "", ylab = "", col = 1, ...)
```

Arguments

x	x coordinates of stick origins.
y	y coordinates of stick origins. If not supplied, 0 will be used; if length is less than that of x, the first number is repeated and the rest are ignored.
u	x component of stick length.
v	y component of stick length.
yscale	scale from u and v to y (see "Description").
add	boolean, set TRUE to add to an existing plot.
length	value to be provided to arrows; here, we set a default that is smaller than normally used, because these plots tend to be crowded in oceanographic applications.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.

`mar`	value to be used with `par("mar")`.
`xlab, ylab`	labels for the plot axes. The default is not to label them.
`col`	colour of sticks, in either numerical or character format. This is made to have length matching that of `x` by a call to `rep`, which can be handy in e.g. colourizing a velocity field by day.
`...`	graphical parameters passed down to `arrows`. It is common, for example, to use smaller arrow heads than `arrows` uses; see "Examples".

Author(s)

Dan Kelley

Examples

```
library(oce)

# Flow from a point source
n <- 16
x <- rep(0, n)
y <- rep(0, n)
theta <- seq(0, 2*pi, length.out=n)
u <- sin(theta)
v <- cos(theta)
plotSticks(x, y, u, v, xlim=c(-2, 2), ylim=c(-2, 2))
rm(n, x, y, theta, u, v)

# Oceanographic example
data(met)
t <- met[["time"]]
u <- met[["u"]]
v <- met[["v"]]
p <- met[["pressure"]]
oce.plot.ts(t, p)
plotSticks(t, 99, u, v, yscale=25, add=TRUE)
```

`plotTaylor` *Plot a Model-data Comparison Diagram*

Description

Creates a diagram as described by Taylor (2001). The graph is in the form of a semicircle, with radial lines and spokes connecting at a focus point on the flat (lower) edge. The radius of a point on the graph indicates the standard deviation of the corresponding quantity, i.e. x and the columns in y. The angle connecting a point on the graph to the focus provides an indication of correlation coefficient with respect to x. The "east" side of the graph indicates $R = 1$, while $R = 0$ is at the "north edge" and $R = -1$ is at the "west" side. The x data are indicated with a bullet on the graph, appearing on the lower edge to the right of the focus at a distance indicating the standard deviation of x. The other points on the graph represent the columns of y, coded automatically or with the supplied values of pch and col. The example shows two tidal models of the Halifax sealevel data, computed with tidem with just the M2 component and the S2 component; the graph indicates that the M2 model is much better than the S2 model.

Usage

```
plotTaylor(x, y, scale, pch, col, labels, pos, ...)
```

Arguments

x	a vector of reference values of some quantity, e.g. measured over time or space.
y	a matrix whose columns hold values of values to be compared with those in x. (If y is a vector, it is converted first to a one-column matrix).
scale	optional scale, interpreted as the maximum value of standard deviation.
pch	list of characters to plot, one for each column of y.
col	list of colours for points on the plot, one for each column of y.
labels	optional vector of strings to use for labelling the points.
pos	optional vector of positions for labelling strings. If not provided, labels will be to the left of the symbols.
...	optional arguments passed by plotTaylor to more child functions.

Author(s)

Dan Kelley

References

Taylor, Karl E., 2001. Summarizing multiple aspects of model performance in a single diagram, *J. Geophys. Res.*, 106:D7, 7183–7192.

Examples

```
library(oce)
data(sealevel)
x <- sealevel[["elevation"]]
M2 <- predict(tidem(sealevel, constituents="M2"))
S2 <- predict(tidem(sealevel, constituents=c("S2")))
plotTaylor(x, cbind(M2, S2))
```

plotTS *Plot Temperature-Salinity Diagram*

Description

Creates a temperature-salinity plot for a CTD cast, with labeled isopycnals.

Usage

```
plotTS(x, inSitu = FALSE, type = "p", referencePressure = 0,
  nlevels = 6, levels, grid = TRUE, col.grid = "lightgray",
  lty.grid = "dotted", rho1000 = FALSE, eos = getOption("oceEOS", default
  = "gsw"), cex = par("cex"), col = par("col"), pch = par("pch"), bg,
  pt.bg = "transparent", col.rho = "darkgray", cex.rho = 3/4 * par("cex"),
  rotate = TRUE, useSmoothScatter = FALSE, xlab, ylab, Slim, Tlim,
  drawFreezing = TRUE, mgp = getOption("oceMgp"), mar = c(mgp[1] + 1.5,
  mgp[1] + 1.5, mgp[1], mgp[1]), lwd = par("lwd"), lty = par("lty"),
  lwd.rho = par("lwd"), lty.rho = par("lty"), add = FALSE,
  inset = FALSE, debug = getOption("oceDebug"), ...)
```

Arguments

x	A ctd object, i.e. one inheriting from ctd-class.
inSitu	A boolean indicating whether to use in-situ temperature or (the default) potential temperature, calculated with reference pressure given by referencePressure. This is ignored if eos="gsw", because those cases the y axis is necessarily the conservative formulation of temperature.
type	representation of data, "p" for points, "l" for connecting lines, or "n" for no indication.
referencePressure	reference pressure, to be used in calculating potential temperature, if inSitu is FALSE.
nlevels	Number of automatically-selected isopycnal levels (ignored if levels is supplied).
levels	Optional vector of desired isopycnal levels.
grid	a flag that can be set to TRUE to get a grid.
col.grid	colour for grid.
lty.grid	line type for grid.
rho1000	if TRUE, label isopycnals as e.g. 1024; if FALSE, label as e.g. 24
eos	equation of state to be used, either "unesco" or "gsw".
cex	character-expansion factor for symbols, as in par("cex").
col	colour for symbols.
pch	symbol type, as in par("pch").

bg	optional colour to be painted under plotting area, before plotting. (This is useful for cases in which `inset=TRUE`.)
pt.bg	inside colour for symbols with pch in 21:25
col.rho	colour for isopycnal lines.
cex.rho	size of isopycnal labels.
rotate	if TRUE, labels in right-hand margin are written vertically
useSmoothScatter	if TRUE, use `smoothScatter` to plot the points.
xlab	optional label for the x axis, with default "Salinity [PSU]".
ylab	optional label for the y axis, with default "Temperature [C]".
Slim	optional limits for salinity axis, otherwise inferred from data.
Tlim	optional limits for temperature axis, otherwise inferred from data.
drawFreezing	logical indication of whether to draw a freezing-point line. This is based on zero pressure. If `eos="unesco"` then `swTFreeze` is used to compute the curve, whereas if `eos="gsw"` then `gsw_CT_freezing` is used; in each case, zero pressure is used.
mgp	3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with `par("mar")`. If set to `NULL`, then `par("mar")` is used. A good choice for a TS diagram with a palette to the right is `mar=par("mar")+c(0, 0, 0, 1))`.
lwd	line width of lines or symbols.
lty	line type of lines or symbols.
lwd.rho	line width for density curves.
lty.rho	line type for density curves.
add	a flag that controls whether to add to an existing plot. (It makes sense to use `add=TRUE` in the `panel` argument of a `coplot`, for example.)
inset	set to `TRUE` for use within `plotInset`. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by `plotInset`.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Value

A list is silently returned, containing `xat` and `yat`, values that can be used by `oce.grid` to add a grid to the plot.

Author(s)

Dan Kelley

See Also

summary,ctd-method summarizes the information, while read.ctd scans it from a file.

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, tidem-class

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
library(oce)
data(ctd)
plotTS(ctd)
```

---

predict.tidem — *Predict a Time Series from a Tidem Tidal Model*

Description

Predict a time series from a tidal model. This is a wrapper around the predict method for `object$model`.

Usage

```
## S3 method for class 'tidem'
predict(object, newdata, ...)
```

Arguments

`object`	A `tidem` object, i.e. one inheriting from tidem-class.
`newdata`	optional vector of POSIXt times at which to make the prediction. If not present, `predict.tidem` uses the times that were provided in the original call to tidem.
`...`	optional arguments passed on to children.

Value

A vector of predictions.

Author(s)

Dan Kelley

See Also

Other things related to `tidem` data: `[[,tidem-method`, `[[<-,tidem-method`, `plot,tidem-method`, `summary,tidem-method`, `tidedata`, `tidem-class`, `tidemAstron`, `tidemVuf`, `tidem`

Examples

```
## Not run:
library(oce)
# 1. tidal anomaly
data(sealevelTuktoyaktuk)
time <- sealevelTuktoyaktuk[["time"]]
elevation <- sealevelTuktoyaktuk[["elevation"]]
oce.plot.ts(time, elevation, type='l', ylab="Height [m]", ylim=c(-2, 6))
tide <- tidem(sealevelTuktoyaktuk)
lines(time, elevation - predict(tide), col="red")
abline(h=0, col="red")

# 2. prediction at specified times
data(sealevel)
m <- tidem(sealevel)
## Check fit over 2 days (interpolating to finer timescale)
look <- 1:48
time <- sealevel[["time"]]
elevation <- sealevel[["elevation"]]
oce.plot.ts(time[look], elevation[look])
# 360s = 10 minute timescale
t <- seq(from=time[1], to=time[max(look)], by=360)
lines(t, predict(m,newdata=t), col='red')
legend("topright", col=c("black","red"),
legend=c("data","model"),lwd=1)

## End(Not run)
```

---

prettyPosition *Pretty lat/lon in deg, min, sec*

Description

Round a geographical positions in degrees, minutes, and seconds Depending on the range of values in `x`, rounding is done to degrees, half-degrees, minutes, etc.

Usage

```
prettyPosition(x, debug = getOption("oceDebug"))
```

Arguments

x	a series of one or more values of a latitude or longitude, in decimal degrees
debug	set to a positive value to get debugging information during processing.

Value

A vector of numbers that will yield good axis labels if `formatPosition` is used.

Author(s)

Dan Kelley

Examples

```
library(oce)
formatPosition(prettyPosition(10+1:10/60+2.8/3600))
```

---

processingLog<- *Add an item to a processing log (in place)*

Description

Add an item to a processing log (in place)

Usage

```
processingLog(x) <- value
```

Arguments

x	An oce object.
value	A character string with the description of the logged activity.

Examples

```
data(ctd)
processingLogShow(ctd)
processingLog(ctd) <- "test"
processingLogShow(ctd)
```

processingLogAppend *Append an item to a processing log*

Description

Append an item to a processing log

Usage

```
processingLogAppend(h, value = "")
```

Arguments

`h`	either the `processingLog` slot of an object, or an `oce` object from which the processingLog will be extracted
`value`	A string indicating the text of the log entry.

Value

An `list` containing items named `time` and `value`, i.e. the times of entries and the text notations of those entries..

processingLogItem *Create an item that can be inserted into a processing log*

Description

A function is used internally to initialize processing logs. Users will probably prefer to use `processingLogAppend` instead.

Usage

```
processingLogItem(value = "")
```

Arguments

`value`	A string that will be used for the item.k

Value

A `list` containing `time`, which is the `Sys.time` at the moment the function is called and `value`, a string that is set to the argument of the same name.

`processingLogShow` *Show the processing log of an `oce` object*

Description

Show the processing log of an oce object

Usage

```
processingLogShow(x)
```

Arguments

`x`	An oce object.

`pwelch` *Welch periodogram*

Description

Compute periodogram using the Welch (1967) method. First, `x` is broken up into chunks, overlapping as specified by `noverlap`. These chunks are then detrended with `detrend`, multiplied by the window, and then passed to `spectrum`. The resulting spectra are then averaged, with the results being stored in `spec` of the return value. Other entries of the return value mimic those returned by `spectrum`.

Usage

```
pwelch(x, window, noverlap, nfft, fs, spectrumtype, esttype, plot = TRUE,
  debug = getOption("oceDebug"), ...)
```

Arguments

`x`	a vector or timeseries to be analyzed. If a timeseries, then there is no need to specify `fs`.
`window`	window specification, either a single value giving the number of windows to use, or a vector of window coefficients. If not specified, then 8 windows are used, each with a Hamming (raised half-cosine) window.
`noverlap`	number of points to overlap between windows. If not specified, this will be set to half the window length.
`nfft`	length of FFT. This cannot be given if `window` is given, and the latter is a single integer.
`fs`	frequency of time-series. If `x` is a time-series, and if `fs` is supplied, then time-series is altered to have frequency `fs`.

`spectrumtype`	not used (yet)
`esttype`	not used (yet)
`plot`	logical, set to `TRUE` to plot the spectrum.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	optional extra arguments to be passed to `spectrum`. Unless specified in this list, `spectrum` is called with `plot=FALSE` to prevent plotting the separate spectra, and with `taper=0`, which is not needed with the default Hanning window. However, the other defaults of `spectrum` are used, e.g. `detrend=TRUE`.

Value

List mimicking the return value from `spectrum`, containing frequency `freq`, spectral power `spec`, degrees of freedom `df`, bandwidth `bandwidth`, etc.

Bugs

Both bandwidth and degrees of freedom are just copied from the values for one of the chunk spectra, and are thus incorrect. That means the cross indicated on the graph is also incorrect.

Author(s)

Dan Kelley

References

Welch, P. D., 1967. The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms. *IEEE Transactions on Audio Electroacoustics*, AU-15, 70–73.

Examples

```
library(oce)
Fs <- 1000
t <- seq(0, 0.296, 1/Fs)
x <- cos(2 * pi * t * 200) + rnorm(n=length(t))
xts <- ts(x, frequency=Fs)
s <- spectrum(xts, spans=c(3,2), main="random + 200 Hz", log='no')
w <- pwelch(xts, plot=FALSE)
lines(w$freq, w$spec, col="red")
w2 <- pwelch(xts, nfft=75, plot=FALSE)
lines(w2$freq, w2$spec, col='green')
abline(v=200, col="blue", lty="dotted")
cat("Checking spectral levels with Parseval's theorem:\n")
cat("var(x)                              = ", var(x), "\n")
cat("2 * sum(s$spec) * diff(s$freq[1:2]) = ", 2 * sum(s$spec) * diff(s$freq[1:2]), "\n")
cat("sum(w$spec) * diff(s$freq[1:2])     = ", sum(w$spec) * diff(w$freq[1:2]), "\n")
cat("sum(w2$spec) * diff(s$freq[1:2])    = ", sum(w2$spec) * diff(w2$freq[1:2]), "\n")
## co2
```

```
par(mar=c(3,3,2,1), mgp=c(2,0.7,0))
s <- spectrum(co2, plot=FALSE)
plot(log10(s$freq), s$spec * s$freq,
     xlab=expression(log[10]*Frequency), ylab="Power*Frequency", type='l')
title("Variance-preserving spectrum")
pw <- pwelch(co2, nfft=256, plot=FALSE)
lines(log10(pw$freq), pw$spec * pw$freq, col='red')
```

---

rangeExtended — *Calculate Range, Extended a Little, as is Done for Axes*

Description

This is analogous to what is done as part of the R axis range calculation, in the case where `xaxs="r"`.

Usage

```
rangeExtended(x, extend = 0.04)
```

Arguments

`x`	a numeric vector.
`extend`	fraction to extend on either end

Value

A two-element vector with the extended range of `x`.

Author(s)

Dan Kelley

---

rangeLimit — *Substitute NA for data outside a range*

Description

Substitute NA for data outside a range, e.g. to remove wild spikes in data.

Usage

```
rangeLimit(x, min, max)
```

Arguments

`x`	vector of values
`min`	minimum acceptable value. If not supplied, and if `max` is also not supplied, a `min` of the 0.5 percentile will be used.
`max`	maximum acceptable value. If not supplied, and if `min` is also not supplied, a `min` of the 0.995 percentile will be used.

Author(s)

Dan Kelley

Examples

```
ten.to.twenty <- rangeLimit(1:100, 10, 20)
```

`read.ad2cp` *Read an AD2CP File*

Description

This function, introduced in April 2017, is just a temporary interface for separate interpretation code that lives in the 1219 issue directory. THIS IS ONLY FOR DEVELOPERS.

Usage

```
read.ad2cp(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  longitude = NA, latitude = NA, orientation, distance, monitor = FALSE,
  despike = FALSE, processingLog, debug = getOption("oceDebug"), ...)
```

Arguments

`file`	a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
`from`	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
`to`	an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.

`by`	an optional indication of the stride length to use while walking through the file. If this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read, e.g. the default `by=1` means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see "Memory considerations".)
`tz`	character string indicating time zone to be assumed in the data.
`longitude`	optional signed number indicating the longitude in degrees East.
`latitude`	optional signed number indicating the latitude in degrees North.
`orientation`	Optional character string specifying the orientation of the sensor, provided for those cases in which it cannot be inferred from the data file. The valid choices are `"upward"`, `"downward"`, and `"sideward"`.
`distance`	Optional vector holding the distances of bin centres from the sensor. This argument is ignored except for Nortek profilers, and need not be given if the function determines the distances correctly from the data. The problem is that the distance is poorly documented in the Nortek System Integrator Guide (2008 edition, page 31), so the function must rely on word-of-mouth formulae that do not work in all cases.
`monitor`	boolean, set to `TRUE` to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with `txtProgressBar`.
`despike`	if `TRUE`, `despike` will be used to clean anomalous spikes in heading, etc.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	additional arguments, passed to called routines.

Value

An `adp` object, i.e. one inheriting from `adp-class`.

Author(s)

Dan Kelley

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

Examples

```
## Not run:
f <- "/Users/kelley/Dropbox/oce_ad2cp/labtestsig3.ad2cp"
d <- read.ad2cp(f, 1, 10, 1)

## End(Not run)
```

---

read.adp *Read an ADP File*

Description

Read an ADP data file, producing an adp object, i.e. one inheriting from adp-class.

Usage

```
read.adp(file, from, to, by, tz = getOption("oceTz"), longitude = NA,
  latitude = NA, manufacturer = c("rdi", "nortek", "sontek"),
  monitor = FALSE, despike = FALSE, processingLog,
  debug = getOption("oceDebug"), ...)
```

Arguments

file	a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
from	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
to	an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.
by	an optional indication of the stride length to use while walking through the file. If this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read, e.g. the default `by=1` means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see "Memory considerations".)
tz	character string indicating time zone to be assumed in the data.
longitude	optional signed number indicating the longitude in degrees East.
latitude	optional signed number indicating the latitude in degrees North.
manufacturer	a character string indicating the manufacturer, used by the general function `read.adp` to select a subsidiary function to use, such as `read.adp.nortek`.

`monitor`	boolean, set to `TRUE` to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with `txtProgressBar`.
`despike`	if `TRUE`, `despike` will be used to clean anomalous spikes in heading, etc.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	additional arguments, passed to called routines.

Details

Several file types can be handled. Some of these functions are wrappers that map to device names, e.g. `read.aquadoppProfiler` does its work by calling `read.adp.nortek`; in this context, it is worth noting that the "aquadopp" instrument is a one-cell profiler that might just as well have been documented under the heading `read.adv`.

Value

An adp object, i.e. one inheriting from `adp-class`.

Author(s)

Dan Kelley and Clark Richards

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

`read.adp.nortek` *Read a Nortek ADP File*

Description

Read a Nortek ADP File

Usage

```
read.adp.nortek(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  longitude = NA, latitude = NA, type = c("aquadoppHR",
  "aquadoppProfiler", "aquadopp"), orientation, distance, monitor = FALSE,
  despike = FALSE, processingLog, debug = getOption("oceDebug"), ...)
```

Arguments

`file`	a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
`from`	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See “Examples”, and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
`to`	an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.
`by`	an optional indication of the stride length to use while walking through the file. If this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read, e.g. the default `by=1` means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see “Memory considerations”.)
`tz`	character string indicating time zone to be assumed in the data.
`longitude`	optional signed number indicating the longitude in degrees East.
`latitude`	optional signed number indicating the latitude in degrees North.
`type`	a character string indicating the type of instrument.
`orientation`	optional character string specifying the orientation of the sensor, provided for those cases in which it cannot be inferred from the data file. The valid choices are `"upward"`, `"downward"`, and `"sideward"`.
`distance`	optional vector holding the distances of bin centres from the sensor. This argument is ignored except for Nortek profilers, and need not be given if the function determines the distances correctly from the data. The problem is that the distance is poorly documented in the Nortek System Integrator Guide (2008 edition, page 31), so the function must rely on word-of-mouth formulae that do not work in all cases.
`monitor`	boolean, set to `TRUE` to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with `txtProgressBar`.
`despike`	if `TRUE`, `despike` will be used to clean anomalous spikes in heading, etc.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	additional arguments, passed to called routines.

Value

An adp object, i.e. one inheriting from `adp-class`.

Author(s)

Dan Kelley

References

1. Information on Nortek profilers (including the System Integrator Guide, which explains the data format byte-by-byte) is available at http://www.nortekusa.com/. (One must join the site to see the manuals.)

2. The Nortek Knowledge Center http://www.nortekusa.com/en/knowledge-center may be of help if problems arise in dealing with data from Nortek instruments.

See Also

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

---

read.adp.rdi *Read a Teledyne/RDI ADP File*

Description

Read a Teledyne/RDI ADCP file (called 'adp' in oce).

Usage

```
read.adp.rdi(file, from, to, by, tz = getOption("oceTz"), longitude = NA,
  latitude = NA, type = c("workhorse"), monitor = FALSE,
  despike = FALSE, processingLog, testing = FALSE,
  debug = getOption("oceDebug"), ...)
```

Arguments

`file`	a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
`from`	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
`to`	an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.

by	an optional indication of the stride length to use while walking through the file. If this is an integer, then by-1 profiles are skipped between each pair of profiles that is read, e.g. the default by=1 means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see "Memory considerations".)
tz	character string indicating time zone to be assumed in the data.
longitude	optional signed number indicating the longitude in degrees East.
latitude	optional signed number indicating the latitude in degrees North.
type	character string indicating the type of instrument.
monitor	boolean, set to TRUE to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with txtProgressBar.
despike	if TRUE, despike will be used to clean anomalous spikes in heading, etc.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
testing	logical value (IGNORED).
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	additional arguments, passed to called routines.

Details

As of 2016-09-25, this function has provisional functionality to read data from the new "SentinelV" series ADCP – essentially a combination of a 4 beam workhorse with an additional vertical centre beam.

If a heading bias had been set with the EB command during the setup for the deployment, then a heading bias will have been stored in the file's header. This value is stored in the object's metadata as metadata$heading.bias. **Importantly**, this value is subtracted from the headings stored in the file, and the result of this subtraction is stored in the objects heading value (in data$heading). It should be noted that read.adp.rdi() was tested for firmware version 16.30. For other versions, there may be problems. For example, the serial number is not recognized properly for version 16.28.

In Teledyne/RDI ADP data files, velocities are coded to signed 2-byte integers, with a scale factor being used to convert to velocity in metres per second. These two facts control the maximum recordable velocity and the velocity resolution, values that may be retrieved for an ADP object name d with d[["velocityMaximum"]] and d[["velocityResolution"]].

Value

An adp object, i.e. one inheriting from adp-class.

Memory considerations

For RDI files only, and only in the case where by is not specified, an attempt is made to avoid running out of memory by skipping some profiles in large input files. This only applies if from and to are both integers; if they are times, none of the rest of this section applies.

A key issue is that RDI files store velocities in 2-byte values, which is not a format that R supports. These velocities become 8-byte (numeric) values in R. Thus, the R object created by `read.adp.rdi` will require more memory than that of the data file. A scale factor can be estimated by ignoring vector quantities (e.g. time, which has just one value per profile) and concentrating on matrix properties such as velocity, backscatter, and correlation. These three elements have equal dimensions. Thus, each 4-byte slide in the data file (2 bytes + 1 byte + 1 byte) corresponds to 10 bytes in the object (8 bytes + 1 byte + 1 byte). Rounding up the resultant 10/4 to 3 for safety, we conclude that any limit on the size of the R object corresponds to a 3X smaller limit on file size.

Various things can limit the size of objects in R, but a strong upper limit is set by the space the operating system provides to R. The least-performant machines in typical use appear to be Microsoft-Windows systems, which limit R objects to about 2e6 bytes [3]. Since R routinely duplicates objects for certain tasks (e.g. for call-by-value in function evaluation), `read.adp.rdi` uses a safety factor in its calculation of when to auto-decimate a file. This factor is set to 3, based partly on the developers' experience with datasets in their possession. Multiplied by the previously stated safety factor of 3, this suggests that the 2 GB limit on R objects corresponds to approximately a 222 MB limit on file size. In the present version of `read.adp.rdi`, this value is lowered to 200 MB for simplicity. Larger files are considered to be "big", and are decimated unless the user supplies a value for the `by` argument.

The decimation procedure has two cases.

1. *Case 1.* If `from=1` and `to=0` (or if neither `from` or `to` is given), then the intention is to process the full span of the data. If the input file is under 200 MB, then `by` defaults to 1, so that all profiles are read. For larger files, `by` is set to the `ceiling` of the ratio of input file size to 200 MB.
2. *Case 2.* If `from` exceeds 1, and/or `to` is nonzero, then the intention is to process only an interior subset of the file. In this case, `by` is calculated as the `ceiling` of the ratio of `bbp*(1+to-from)` to 200 MB, where `bbp` is the number of file bytes per profile. Of course, `by` is set to 1, if this ratio is less than 1.

If the result of these calculations is that `by` exceeds 1, then messages are printed to alert the user that the file will be decimated, and also `monitor` is set to `TRUE`, so that a textual progress bar is shown.

Author(s)

Dan Kelley and Clark Richards

References

1. Teledyne-RDI, 2007. *WorkHorse commands and output data format.* P/N 957-6156-00 (November 2007). (Section 5.3 h details the binary format, e.g. the file should start with the byte `0x7f` repeated twice, and each profile starts with the bytes `0x80`, followed by `0x00`, followed by the sequence number of the profile, represented as a little-endian two-byte short integer. `read.adp.rdi` uses these sequences to interpret data files.)

2. Teledyne-RDI, 2015. *V Series output data format.* P/N 95D-6022-00 (May 2015).

3. See `Memory-limits` for more on the 2 GB limit for R on windows machines (but note that this documentation erroneously states the unit as Gb, which is typically used for gigabits).

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

---

read.adp.sontek *Read a Sontek ADP File*

Description

Read a Sontek acoustic-Doppler profiler file [1].

Usage

```
read.adp.sontek(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  longitude = NA, latitude = NA, type = c("adp", "pcadp"),
  monitor = FALSE, despike = FALSE, processingLog,
  debug = getOption("oceDebug"), ...)
```

Arguments

`file`	a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
`from`	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
`to`	an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.
`by`	an optional indication of the stride length to use while walking through the file. If this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read, e.g. the default `by=1` means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see "Memory considerations".)
`tz`	character string indicating time zone to be assumed in the data.
`longitude`	optional signed number indicating the longitude in degrees East.
`latitude`	optional signed number indicating the latitude in degrees North.
`type`	A character string indicating the type of instrument.

`monitor`	boolean, set to `TRUE` to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with `txtProgressBar`.
`despike`	if `TRUE`, `despike` will be used to clean anomalous spikes in heading, etc.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	additional arguments, passed to called routines.

Value

An adp object, i.e. one inheriting from `adp-class`.

Author(s)

Dan Kelley and Clark Richards

References

1. Information about Sontek profilers is available at

`http://www.sontek.com`.

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

`read.adp.sontek.serial` *Read a serial Sontek ADP file*

Description

Read a Sontek acoustic-Doppler profiler file, in a serial form that is possibly unique to Dalhousie University.

Usage

```
read.adp.sontek.serial(file, from = 1, to, by = 1,
  tz = getOption("oceTz"), longitude = NA, latitude = NA,
  type = c("adp", "pcadp"), beamAngle = 25, orientation, monitor = FALSE,
  processingLog, debug = getOption("oceDebug"))
```

Arguments

`file`	a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
`from`	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
`to`	an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.
`by`	an optional indication of the stride length to use while walking through the file. If this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read, e.g. the default `by=1` means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see "Memory considerations".)
`tz`	character string indicating time zone to be assumed in the data.
`longitude`	optional signed number indicating the longitude in degrees East.
`latitude`	optional signed number indicating the latitude in degrees North.
`type`	a character string indicating the type of instrument.
`beamAngle`	angle between instrument axis and beams, in degrees.
`orientation`	optional character string specifying the orientation of the sensor, provided for those cases in which it cannot be inferred from the data file. The valid choices are `"upward"`, `"downward"`, and `"sideward"`.
`monitor`	boolean, set to `TRUE` to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with `txtProgressBar`.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	additional arguments, passed to called routines.

Value

An adp object, i.e. one inheriting from `adp-class`.

Author(s)

Dan Kelley and Clark Richards

See Also

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

---

read.adv — *Read an ADV data file*

Description

Read an ADV data file, producing an object of type adv. This function works by transferring control to a more specialized function, e.g. read.adp.nortek and read.adp.sontek, and in many cases users will find it preferable to either use these or the several even more specialized functions, if the file type is known.

Usage

```
read.adv(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  type = c("nortek", "sontek", "sontek.adr", "sontek.text"), header = TRUE,
  longitude = NA, latitude = NA, start = NULL, deltat = NA,
  debug = getOption("oceDebug"), monitor = FALSE, processingLog = NULL)
```

Arguments

file	a connection or a character string giving the name of the file to load. It is also possible to give file as a vector of filenames, to handle the case of data split into files by a data logger. In the multi-file case, header must be FALSE, start must be a vector of times, and deltat must be provided.
from	index number of the first profile to be read, or the time of that profile, as created with as.POSIXct (hint: use tz="UTC"). This argument is ignored if header==FALSE. See “Examples”.
to	indication of the last profile to read, in a format matching that of from. This is ignored if header==FALSE.
by	an indication of the stride length to use while walking through the file. This is ignored if header==FALSE. Otherwise, if this is an integer, then by-1 profiles are skipped between each pair of profiles that is read. This may not make much sense, if the data are not equi-spaced in time. If by is a string representing a time interval, in colon-separated format, then this interval is divided by the sampling interval, to get the stride length. *BUG:* by only partially works; see the “Bugs” section below.
tz	character string indicating time zone to be assumed in the data.
type	character string indicating type of file, and used by read.adv to dispatch to one of the speciality functions.

header	A logical value indicating whether the file starts with a header. (This will not be the case for files that are created by data loggers that chop the raw data up into a series of sub-files, e.g. once per hour.)
longitude	optional signed number indicating the longitude in degrees East.
latitude	optional signed number indicating the latitude in degrees North.
start	the time of the first sample, typically created with `as.POSIXct`. This may be a vector of times, if `filename` is a vector of file names.
deltat	the time between samples. (This is mandatory if `header=FALSE`.)
debug	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies. For example, `read.adv.nortek()` calls `read.header.nortek()`, so that `read.adv.nortek(...,debug=2)` provides information about not just the main body of the data file, but also the details of the header.
monitor	boolean, set to `TRUE` to provide an indication of every data burst read.
processingLog	if provided, the action item to be stored in the log. This parameter is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

Details

Files *without* headers may be created in experiments in which a data logger was set up to monitor the serial data stream from an instrument. The lack of header information places a burden on the user, who must supply such basic information as the times of observations, the instrument orientation, the instrument coordinate system, etc. Example 3 below shows how to deal with such files. Three things should be noted.

1. The user must choose the appropriate `read.adv` variant corresponding to the instrument in question. (This is necessary because `oceMagic`, which is used by the generic `read.oce` routine, cannot determine the type of instrument by examining a file that lacks a header.)
2. The call to the `read` function must include a start time (`start`) and the number of seconds between data (`deltat`), again, because the instrument data stream may lack those things when the device is set to a serial mode. Also, of course, it is necessary to set `header=FALSE` in the function call.
3. Once the file has been read in, the user will be obliged to specify other information, for the object to be well-formed. For example, the `read` function will have no way of knowing the instrument orientation, the coordinate system being used, the transformation matrix to go from `"beam"` to `"xyz"` coordinates, or the instrument heading, pitch, and roll, to go from `"xyz"` coordinates to `"enu"` coordinates. Such things are illustrated in example 3 below.

In ADV data files, velocities are coded to signed 2-byte integers, with a scale factor being used to convert to velocity in metres per second. These two facts control the maximum recordable velocity and the velocity resolution, values that may be retrieved for an ADV object name `d` with `d[["velocityMaximum"]]` and `d[["velocityResolution"]]`.

Value

An object of adv-class that contains measurements made with an ADV device.

The metadata contains information as given in the following table. The "Nortek name" is the name used in the Nortek System Integrator Guide [reference 1] and the "Sontek name" is the name used in the relevant Sontek documentation. References are given in square brackets.

metadata **name**	**Nortek name**	**Sontek name**	**Meaning**
manufacturer	-	-	Either "nortek" or "sontek"
instrumentType	-	-	Either "vector" or "adv"
filename	-	-	Name of data file(s)
latitude	-	-	Latitude of mooring (if applicable)
longitude	-	-	Longitude of mooring (if applicable)
numberOfSamples	-	-	Number of data samples in file
numberOfBeams	NBeams [1 p18]	-	Number of beams (always 3)
numberOfBeamSequencesPerBurst	NPings	-	number of beam sequences per burst
measurementInterval	MeasInterval [1 p31]	-	
samplingRate	512/(AvgInterval) [1 p30; 4]	-	

The data list contains items with names corresponding to adp objects, with an exception for Nortek data. Nortek instruments report some things at a time interval that is longer than the velocity sampling, and these are stored in data as timeSlow, headingSlow, pitchSlow, rollSlow, and temperatureSlow; if burst sampling was used, there will also be items recordsBurst and timeBurst.

The processingLog is in the standard format.

Nortek files

Sampling-rate and similar issues

The data format is inferred from the System Integrator Guide [1A] and System Integrator Manual [1B]. These document lacks clarity in spots, and so read.adv.nortek contains some assumptions that are noted here, so that users will be aware of possible problems.

A prominent example is the specification of the sampling rate, stored in metadata$sampingRate in the return value. Repeated examination of the System Integrator Guide [1] failed to indicate where this value is stored in the various headers contained in Vector datasets. After some experimentation with a few data files, read.adv.nortek was set up to calculate metadata$samplingRate as 512/AvgInterval where AvgInterval is a part of the "User Configuration" header [1 p30], where the explanation is "average interval in seconds"). This formula was developed through trial and error, but it was confirmed later on the Nortek discussion group, and it should appear in upcoming versions of [1].

Another basic issue is the determination of whether an instrument had recorded in continuous mode or burst mode. One might infer that TimCtrlReg in the "User Configuration" header [1 p30] determines this, in bits 1 and 2. However, this was the case in test files available to the author. For this reason, read.adv.nortek infers the mode by reverse engineering of data files of known configuration. The present version of read.adv.nortek determines the sampling mode from the "NRecords" item of the "Vector Velocity Data" header, which seems to be 0 for data collected continuously, and non-zero for data collected in bursts.

Taking these things together, we come upon the issue of how to infer sampling times for Nortek instruments. There do not seem to be definitive documents on this, and so `read.adv.nortek` is based partly on information (of unknown quality) found on Nortek discusson boards. The present version of `read.adv.nortek` infers the times of velocity observations differently, depending on whether the instrument was set to record in burst mode or continuous mode. For burst mode, times stated in the burst headers are used, but for continous mode, times stated in the "vector system data" are used. On the advice found on a Nortek discussion board, the burst-mode times are offset by 2 seconds to allow for the instrument warm-up period.

Handling IMU (inertial measurement unit) data

Starting in March 2016, `read.adv.nortek` has offered some support for handling IMU (inertial measurement unit) data incorporated into Nortek binary files. This is not described in the Nortek document named "System Integrator Guide" (2008 [1A]) but it appeared in "System Integrator Manual" (2014 [1B]; 2016 [1C]). Confusingly, 1B described 3 varieties of data, whereas 1C does not describe any of these, but describes instead a fourth variety. As of March 2016, `read.adv.nortek` handles all 4 varieties, because files in the various schemes appear to exist. In `oce`, the varieties are named after the byte code that flags them. (Variety `c3` is the one described in [1C]; the others were described in [1B].) The variety is stored in the `metadata` slot of the returned object as a string named `IMUtype`.

For each variety, the reader is cautioned that strong tests have not been performed on the code. One way to test the code is to compare with textual data files produced by the Nortek software. In March 2016, an `oce` user shared a dataset of the `c3` variety, and this permitted detailed comparison between the text file and the values inferred by `read.adv.nortek`. The test suggested agreement (to within the resolution printed in the text file) for velocity (`v` in the `data` slot), signal amplitude (`a`), correlation (`q`), pressure (`p`), the three components of IMU delta angle (`IMUdeltaAngleX` etc), and all components of the rotation matrix (`IMUrotation`). However, the delta velocity signals did not match, with `IMUdeltaVelocityX` disagreeing in the second decimal place, `IMUdeltaVelocityY` component disagreeing in the first, and `IMUdeltaVelocityZ` being out by a factor of about 10. This is github issue 893 (https://github.com/dankelley/oce/issues/893).

- Variety `c3` (signalled by byte 5 of a sequence being `0xc3`) provides information on what Nortek calls DeltaAngle, DeltaVelocity and Orientation Matrix. (Apart from the orientation matrix, Nortek provides no documentation on what these quantities mean.) In the object returned by `read.adv.nortek`, these are stored in the `data` slot as `IMUdeltaAngleX`, `IMUdeltaAngleY`, `IMUdeltaAngleZ`, `IMUdeltaVelocityX`, `IMUdeltaVelocityY`, `IMUdeltaVelocityZ`, and `IMUrotation`, all vectors except the last, which is a 3D array. In addition to these, `IMUtimestamp` is a timestamp, which is not defined in the Nortek documents but seems, from IMU documents [5], to be defined based on a clock that ticks once per 16 microseconds. Caution may be required in dealing with this timestamp, since it seemed sensible in one test case (variety `d3`) but kept reseting to zero in another (variety `c3`). The lack of Nortek documentation on most of these quantities is a roadblock to implementing `oce` functions dealing with IMU-enabled datasets
- Variety `cc` (signalled by byte 5 of a sequence being `0xcc`) provides information on acceleration, angular rotation rate, magnetic vector and orientation matrix. Each is a timeseries. Acceleration is stored in the `data` slot as `IMUaccelX`, `IMUaccelY`, `IMUaccelz`. The angular rotation components are `IMUngrtX`, `IMUngrtY` and `IMUngrtz`. The magnetic data are in `IMUmagrtx`, `IMUmagrty` and `IMUmagrtz`. Finally, `IMUmatrix` is a rotation matrix made up from elements named `M11`, `M12`, etc in the Nortek documentation. In addition to all of these, `IMUtime` stores time in seconds (with an origin whose definition is not stated in [1B]).

- Variety d2 (signalled by byte 5 being 0xd2) provides information on gyro-stabilized acceleration, angular rate and magnetometer vectors. The data stored MUaccelX, IMUangrtX, IMUmagrtX, with similar for Y and Z. Again, time is in IMUtime. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.
- Variety d3 (signalled by byte 5 being 0xd3) provides information on DeltaAngle, DeltaVelocity and magnetometer vectors, stored in IMUdeltaAngleX, IMUdeltaVelocityX, and IMUdeltaMagVectorX, with similar for Y and Z. Again, time is in IMUtime. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.

Author(s)

Dan Kelley

References

1A. Nortek AS. System Integrator Guide (paradopp family of products). June 2008. (Doc No: PSI00-0101-0608). (Users may find it helpful to also examine newer versions of the guide.)

1B. Nortek AS. System Integrator Manual. Dec 2014. (system-integrator-manual_Dec2014_jan.pdf)

1C. Nortek AS. System Integrator Manual. March 2016. (system-integrator-manual_Mar2016.pdf)

2. SonTek/YSI ADVField/Hydra Acoustic Doppler Velocimeter (Field) Technical Documentation (Sept 1, 2001).

3. Appendix 2.2.3 of the Sontek ADV operation Manual, Firmware Version 4.0 (Oct 1997).

4. Nortek Knowledge Center http://www.nortekusa.com/en/knowledge-center

5. A document describing an IMU unit that seems to be close to the one named in [1B,C] as being an adjunct to Nortek Vector systems is at http://files.microstrain.com/3DM-GX3-35-Data-Communications-Protocol.

See Also

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv-class, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

Examples

```
## Not run:
library(oce)
# A nortek Vector file
d <- read.oce("/data/archive/sleiwex/2008/moorings/m05/adv/nortek_1943/raw/adv_nortek_1943.vec",
              from=as.POSIXct("2008-06-26 00:00:00", tz="UTC"),
              to=as.POSIXct("2008-06-26 00:00:10", tz="UTC"))
plot(d, which=c(1:3,15))

## End(Not run)
```

`read.adv.nortek` *Read an ADV data file*

Description

Read an ADV data file, producing an object of type adv. This function works by transferring control to a more specialized function, e.g. `read.adp.nortek` and `read.adp.sontek`, and in many cases users will find it preferable to either use these or the several even more specialized functions, if the file type is known.

Usage

```
read.adv.nortek(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  header = TRUE, longitude = NA, latitude = NA, type = c("vector",
  "aquadopp"), haveAnalog1 = FALSE, haveAnalog2 = FALSE,
  debug = getOption("oceDebug"), monitor = FALSE, processingLog = NULL)
```

Arguments

`file`	a connection or a character string giving the name of the file to load. It is also possible to give `file` as a vector of filenames, to handle the case of data split into files by a data logger. In the multi-file case, `header` must be `FALSE`, `start` must be a vector of times, and `deltat` must be provided.
`from`	index number of the first profile to be read, or the time of that profile, as created with `as.POSIXct` (hint: use `tz="UTC"`). This argument is ignored if `header==FALSE`. See "Examples".
`to`	indication of the last profile to read, in a format matching that of `from`. This is ignored if `header==FALSE`.
`by`	an indication of the stride length to use while walking through the file. This is ignored if `header==FALSE`. Otherwise, if this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read. This may not make much sense, if the data are not equi-spaced in time. If by is a string representing a time interval, in colon-separated format, then this interval is divided by the sampling interval, to get the stride length. *BUG:* by only partially works; see the "Bugs" section below.
`tz`	character string indicating time zone to be assumed in the data.
`header`	A logical value indicating whether the file starts with a header. (This will not be the case for files that are created by data loggers that chop the raw data up into a series of sub-files, e.g. once per hour.)
`longitude`	optional signed number indicating the longitude in degrees East.
`latitude`	optional signed number indicating the latitude in degrees North.
`type`	A string indicating which type of Nortek device produced the data file, `vector` or `aquadopp`.
`haveAnalog1`	A logical value indicating whether the data file has 'analog1' data.

`haveAnalog2`	A logical value indicating whether the data file has 'analog2' data.
`debug`	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies. For example, `read.adv.nortek()` calls `read.header.nortek()`, so that `read.adv.nortek(...,debug=2)` provides information about not just the main body of the data file, but also the details of the header.
`monitor`	boolean, set to `TRUE` to provide an indication of every data burst read.
`processingLog`	if provided, the action item to be stored in the log. This parameter is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

Details

Files *without* headers may be created in experiments in which a data logger was set up to monitor the serial data stream from an instrument. The lack of header information places a burden on the user, who must supply such basic information as the times of observations, the instrument orientation, the instrument coordinate system, etc. Example 3 below shows how to deal with such files. Three things should be noted.

1. The user must choose the appropriate `read.adv` variant corresponding to the instrument in question. (This is necessary because `oceMagic`, which is used by the generic `read.oce` routine, cannot determine the type of instrument by examining a file that lacks a header.)
2. The call to the `read` function must include a start time (`start`) and the number of seconds between data (`deltat`), again, because the instrument data stream may lack those things when the device is set to a serial mode. Also, of course, it is necessary to set `header=FALSE` in the function call.
3. Once the file has been read in, the user will be obliged to specify other information, for the object to be well-formed. For example, the `read` function will have no way of knowing the instrument orientation, the coordinate system being used, the transformation matrix to go from `"beam"` to `"xyz"` coordinates, or the instrument heading, pitch, and roll, to go from `"xyz"` coordinates to `"enu"` coordinates. Such things are illustrated in example 3 below.

In ADV data files, velocities are coded to signed 2-byte integers, with a scale factor being used to convert to velocity in metres per second. These two facts control the maximum recordable velocity and the velocity resolution, values that may be retrieved for an ADV object name `d` with `d[["velocityMaximum"]]` and `d[["velocityResolution"]]`.

Value

An object of `adv-class` that contains measurements made with an ADV device.

The `metadata` contains information as given in the following table. The "Nortek name" is the name used in the Nortek System Integrator Guide [reference 1] and the "Sontek name" is the name used in the relevant Sontek documentation. References are given in square brackets.

`metadata` **name**	**Nortek name**	**Sontek name**	**Meaning**
`manufacturer`	-	-	Either `"nortek"` or `"sontek"`
`instrumentType`	-	-	Either `"vector"` or `"adv"`
`filename`	-	-	Name of data file(s)

`latitude`	-	-	Latitude of mooring (if applicable)
`longitude`	-	-	Longitude of mooring (if applicable)
`numberOfSamples`	-	-	Number of data samples in file
`numberOfBeams`	NBeams [1 p18]	-	Number of beams (always 3)
`numberOfBeamSequencesPerBurst`	NPings	-	number of beam sequences per burst
`measurementInterval`	MeasInterval [1 p31]	-	
`samplingRate`	512/(AvgInterval) [1 p30; 4]	-	

The `data` list contains items with names corresponding to `adp` objects, with an exception for Nortek data. Nortek instruments report some things at a time interval that is longer than the velocity sampling, and these are stored in `data` as `timeSlow`, `headingSlow`, `pitchSlow`, `rollSlow`, and `temperatureSlow`; if burst sampling was used, there will also be items `recordsBurst` and `timeBurst`.

The `processingLog` is in the standard format.

Nortek files

Sampling-rate and similar issues

The data format is inferred from the System Integrator Guide [1A] and System Integrator Manual [1B]. These document lacks clarity in spots, and so `read.adv.nortek` contains some assumptions that are noted here, so that users will be aware of possible problems.

A prominent example is the specification of the sampling rate, stored in `metadata$sampingRate` in the return value. Repeated examination of the System Integrator Guide [1] failed to indicate where this value is stored in the various headers contained in Vector datasets. After some experimentation with a few data files, `read.adv.nortek` was set up to calculate `metadata$samplingRate` as `512/AvgInterval` where `AvgInterval` is a part of the "User Configuration" header [1 p30], where the explanation is "average interval in seconds"). This formula was developed through trial and error, but it was confirmed later on the Nortek discussion group, and it should appear in upcoming versions of [1].

Another basic issue is the determination of whether an instrument had recorded in continuous mode or burst mode. One might infer that `TimCtrlReg` in the "User Configuration" header [1 p30] determines this, in bits 1 and 2. However, this was the case in test files available to the author. For this reason, `read.adv.nortek` infers the mode by reverse engineering of data files of known configuration. The present version of `read.adv.nortek` determines the sampling mode from the "`NRecords`" item of the "Vector Velocity Data" header, which seems to be 0 for data collected continuously, and non-zero for data collected in bursts.

Taking these things together, we come upon the issue of how to infer sampling times for Nortek instruments. There do not seem to be definitive documents on this, and so `read.adv.nortek` is based partly on information (of unknown quality) found on Nortek discusson boards. The present version of `read.adv.nortek` infers the times of velocity observations differently, depending on whether the instrument was set to record in burst mode or continuous mode. For burst mode, times stated in the burst headers are used, but for continous mode, times stated in the "vector system data" are used. On the advice found on a Nortek discussion board, the burst-mode times are offset by 2 seconds to allow for the instrument warm-up period.

Handling IMU (inertial measurement unit) data

Starting in March 2016, `read.adv.nortek` has offered some support for handling IMU (inertial measurement unit) data incorporated into Nortek binary files. This is not described in the Nortek document named “System Integrator Guide” (2008 [1A]) but it appeared in “System Integrator Manual” (2014 [1B]; 2016 [1C]). Confusingly, 1B described 3 varieties of data, whereas 1C does not describe any of these, but describes instead a fourth variety. As of March 2016, `read.adv.nortek` handles all 4 varieties, because files in the various schemes appear to exist. In `oce`, the varieties are named after the byte code that flags them. (Variety `c3` is the one described in [1C]; the others were described in [1B].) The variety is stored in the `metadata` slot of the returned object as a string named `IMUtype`.

For each variety, the reader is cautioned that strong tests have not been performed on the code. One way to test the code is to compare with textual data files produced by the Nortek software. In March 2016, an `oce` user shared a dataset of the `c3` variety, and this permitted detailed comparison between the text file and the values inferred by `read.adv.nortek`. The test suggested agreement (to within the resolution printed in the text file) for velocity (`v` in the `data` slot), signal amplitude (`a`), correlation (`q`), pressure (`p`), the three components of IMU delta angle (`IMUdeltaAngleX` etc), and all components of the rotation matrix (`IMUrotation`). However, the delta velocity signals did not match, with `IMUdeltaVelocityX` disagreeing in the second decimal place, `IMUdeltaVelocityY` component disagreeing in the first, and `IMUdeltaVelocityZ` being out by a factor of about 10. This is github issue 893 (https://github.com/dankelley/oce/issues/893).

- Variety `c3` (signalled by byte 5 of a sequence being `0xc3`) provides information on what Nortek calls DeltaAngle, DeltaVelocity and Orientation Matrix. (Apart from the orientation matrix, Nortek provides no documentation on what these quantities mean.) In the object returned by `read.adv.nortek`, these are stored in the `data` slot as `IMUdeltaAngleX`, `IMUdeltaAngleY`, `IMUdeltaAngleZ`, `IMUdeltaVelocityX`, `IMUdeltaVelocityY`, `IMUdeltaVelocityZ`, and `IMUrotation`, all vectors except the last, which is a 3D array. In addition to these, `IMUtimestamp` is a timestamp, which is not defined in the Nortek documents but seems, from IMU documents [5], to be defined based on a clock that ticks once per 16 microseconds. Caution may be required in dealing with this timestamp, since it seemed sensible in one test case (variety `d3`) but kept reseting to zero in another (variety `c3`). The lack of Nortek documentation on most of these quantities is a roadblock to implementing `oce` functions dealing with IMU-enabled datasets
- Variety `cc` (signalled by byte 5 of a sequence being `0xcc`) provides information on acceleration, angular rotation rate, magnetic vector and orientation matrix. Each is a timeseries. Acceleration is stored in the `data` slot as `IMUaccelX`, `IMUaccelY`, `IMUaccelz`. The angular rotation components are `IMUngrtX`, `IMUngrtY` and `IMUngrtz`. The magnetic data are in `IMUmagrtx`, `IMUmagrty` and `IMUmagrtz`. Finally, `IMUmatrix` is a rotation matrix made up from elements named `M11`, `M12`, etc in the Nortek documentation. In addition to all of these, `IMUtime` stores time in seconds (with an origin whose definition is not stated in [1B]).
- Variety `d2` (signalled by byte 5 being `0xd2`) provides information on gyro-stabilized acceleration, angular rate and magnetometer vectors. The data stored `MUaccelX`, `IMUangrtX`, `IMUmagrtX`, with similar for `Y` and `Z`. Again, time is in `IMUtime`. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.
- Variety `d3` (signalled by byte 5 being `0xd3`) provides information on DeltaAngle, DeltaVelocity and magnetometer vectors, stored in `IMUdeltaAngleX`, `IMUdeltaVelocityX`, and `IMUdeltaMagVectorX`, with similar for `Y` and `Z`. Again, time is in `IMUtime`. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.

Author(s)

Dan Kelley

References

1A. Nortek AS. System Integrator Guide (paradopp family of products). June 2008. (Doc No: PSI00-0101-0608). (Users may find it helpful to also examine newer versions of the guide.)

1B. Nortek AS. System Integrator Manual. Dec 2014. (`system-integrator-manual_Dec2014_jan.pdf`)

1C. Nortek AS. System Integrator Manual. March 2016. (`system-integrator-manual_Mar2016.pdf`)

2. SonTek/YSI ADVField/Hydra Acoustic Doppler Velocimeter (Field) Technical Documentation (Sept 1, 2001).

3. Appendix 2.2.3 of the Sontek ADV operation Manual, Firmware Version 4.0 (Oct 1997).

4. Nortek Knowledge Center `http://www.nortekusa.com/en/knowledge-center`

5. A document describing an IMU unit that seems to be close to the one named in [1B,C] as being an adjunct to Nortek Vector systems is at `http://files.microstrain.com/3DM-GX3-35-Data-Communications-Protocol.`

See Also

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOtherAdv`, `enuToOther`, `plot,adv-method`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnuAdv`, `xyzToEnu`

Examples

```
## Not run:
library(oce)
# A nortek Vector file
d <- read.oce("/data/archive/sleiwex/2008/moorings/m05/adv/nortek_1943/raw/adv_nortek_1943.vec",
              from=as.POSIXct("2008-06-26 00:00:00", tz="UTC"),
              to=as.POSIXct("2008-06-26 00:00:10", tz="UTC"))
plot(d, which=c(1:3,15))

## End(Not run)
```

`read.adv.sontek.adr` *Read an ADV data file*

Description

Read an ADV data file, producing an object of type adv. This function works by transferring control to a more specialized function, e.g. `read.adp.nortek` and `read.adp.sontek`, and in many cases users will find it preferable to either use these or the several even more specialized functions, if the file type is known.

Usage

```
read.adv.sontek.adr(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  header = TRUE, longitude = NA, latitude = NA,
  debug = getOption("oceDebug"), monitor = FALSE, processingLog = NULL)
```

Arguments

- `file` — a connection or a character string giving the name of the file to load. It is also possible to give `file` as a vector of filenames, to handle the case of data split into files by a data logger. In the multi-file case, `header` must be `FALSE`, `start` must be a vector of times, and `deltat` must be provided.
- `from` — index number of the first profile to be read, or the time of that profile, as created with `as.POSIXct` (hint: use `tz="UTC"`). This argument is ignored if `header==FALSE`. See "Examples".
- `to` — indication of the last profile to read, in a format matching that of `from`. This is ignored if `header==FALSE`.
- `by` — an indication of the stride length to use while walking through the file. This is ignored if `header==FALSE`. Otherwise, if this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read. This may not make much sense, if the data are not equi-spaced in time. If by is a string representing a time interval, in colon-separated format, then this interval is divided by the sampling interval, to get the stride length. *BUG:* by only partially works; see the "Bugs" section below.
- `tz` — character string indicating time zone to be assumed in the data.
- `header` — A logical value indicating whether the file starts with a header. (This will not be the case for files that are created by data loggers that chop the raw data up into a series of sub-files, e.g. once per hour.)
- `longitude` — optional signed number indicating the longitude in degrees East.
- `latitude` — optional signed number indicating the latitude in degrees North.
- `debug` — a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies. For example, `read.adv.nortek()` calls `read.header.nortek()`, so that `read.adv.nortek(...,debug=2)` provides information about not just the main body of the data file, but also the details of the header.
- `monitor` — boolean, set to `TRUE` to provide an indication of every data burst read.
- `processingLog` — if provided, the action item to be stored in the log. This parameter is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

Details

Files *without* headers may be created in experiments in which a data logger was set up to monitor the serial data stream from an instrument. The lack of header information places a burden on the user, who must supply such basic information as the times of observations, the instrument orientation, the instrument coordinate system, etc. Example 3 below shows how to deal with such files. Three things should be noted.

1. The user must choose the appropriate `read.adv` variant corresponding to the instrument in question. (This is necessary because `oceMagic`, which is used by the generic `read.oce` routine, cannot determine the type of instrument by examining a file that lacks a header.)
2. The call to the `read` function must include a start time (`start`) and the number of seconds between data (`deltat`), again, because the instrument data stream may lack those things when the device is set to a serial mode. Also, of course, it is necessary to set `header=FALSE` in the function call.
3. Once the file has been read in, the user will be obliged to specify other information, for the object to be well-formed. For example, the `read` function will have no way of knowing the instrument orientation, the coordinate system being used, the transformation matrix to go from `"beam"` to `"xyz"` coordinates, or the instrument heading, pitch, and roll, to go from `"xyz"` coordinates to `"enu"` coordinates. Such things are illustrated in example 3 below.

In ADV data files, velocities are coded to signed 2-byte integers, with a scale factor being used to convert to velocity in metres per second. These two facts control the maximum recordable velocity and the velocity resolution, values that may be retrieved for an ADV object name `d` with `d[["velocityMaximum"]]` and `d[["velocityResolution"]]`.

Value

An object of `adv-class` that contains measurements made with an ADV device.

The `metadata` contains information as given in the following table. The “Nortek name” is the name used in the Nortek System Integrator Guide [reference 1] and the “Sontek name” is the name used in the relevant Sontek documentation. References are given in square brackets.

`metadata` **name**	**Nortek name**	**Sontek name**	**Meaning**
`manufacturer`	-	-	Either `"nortek"` or `"sontek"`
`instrumentType`	-	-	Either `"vector"` or `"adv"`
`filename`	-	-	Name of data file(s)
`latitude`	-	-	Latitude of mooring (if applicable)
`longitude`	-	-	Longitude of mooring (if applicable)
`numberOfSamples`	-	-	Number of data samples in file
`numberOfBeams`	NBeams [1 p18]	-	Number of beams (always 3)
`numberOfBeamSequencesPerBurst`	NPings	-	number of beam sequences per burst
`measurementInterval`	MeasInterval [1 p31]	-	
`samplingRate`	512/(AvgInterval) [1 p30; 4]	-	

The `data` list contains items with names corresponding to `adp` objects, with an exception for Nortek data. Nortek instruments report some things at a time interval that is longer than the velocity sampling, and these are stored in `data` as `timeSlow`, `headingSlow`, `pitchSlow`, `rollSlow`, and `temperatureSlow`; if burst sampling was used, there will also be items `recordsBurst` and `timeBurst`.

The `processingLog` is in the standard format.

Nortek files

Sampling-rate and similar issues

The data format is inferred from the System Integrator Guide [1A] and System Integrator Manual [1B]. These document lacks clarity in spots, and so `read.adv.nortek` contains some assumptions that are noted here, so that users will be aware of possible problems.

A prominent example is the specification of the sampling rate, stored in `metadata$sampingRate` in the return value. Repeated examination of the System Integrator Guide [1] failed to indicate where this value is stored in the various headers contained in Vector datasets. After some experimentation with a few data files, `read.adv.nortek` was set up to calculate `metadata$samplingRate` as `512/AvgInterval` where `AvgInterval` is a part of the "User Configuration" header [1 p30], where the explanation is "average interval in seconds"). This formula was developed through trial and error, but it was confirmed later on the Nortek discussion group, and it should appear in upcoming versions of [1].

Another basic issue is the determination of whether an instrument had recorded in continuous mode or burst mode. One might infer that `TimCtrlReg` in the "User Configuration" header [1 p30] determines this, in bits 1 and 2. However, this was the case in test files available to the author. For this reason, `read.adv.nortek` infers the mode by reverse engineering of data files of known configuration. The present version of `read.adv.nortek` determines the sampling mode from the "`NRecords`" item of the "Vector Velocity Data" header, which seems to be 0 for data collected continuously, and non-zero for data collected in bursts.

Taking these things together, we come upon the issue of how to infer sampling times for Nortek instruments. There do not seem to be definitive documents on this, and so `read.adv.nortek` is based partly on information (of unknown quality) found on Nortek discusson boards. The present version of `read.adv.nortek` infers the times of velocity observations differently, depending on whether the instrument was set to record in burst mode or continuous mode. For burst mode, times stated in the burst headers are used, but for continous mode, times stated in the "vector system data" are used. On the advice found on a Nortek discussion board, the burst-mode times are offset by 2 seconds to allow for the instrument warm-up period.

Handling IMU (inertial measurement unit) data

Starting in March 2016, `read.adv.nortek` has offered some support for handling IMU (inertial measurement unit) data incorporated into Nortek binary files. This is not described in the Nortek document named "System Integrator Guide" (2008 [1A]) but it appeared in "System Integrator Manual" (2014 [1B]; 2016 [1C]). Confusingly, 1B described 3 varieties of data, whereas 1C does not describe any of these, but describes instead a fourth variety. As of March 2016, `read.adv.nortek` handles all 4 varieties, because files in the various schemes appear to exist. In `oce`, the varieties are named after the byte code that flags them. (Variety `c3` is the one described in [1C]; the others were described in [1B].) The variety is stored in the `metadata` slot of the returned object as a string named `IMUtype`.

For each variety, the reader is cautioned that strong tests have not been performed on the code. One way to test the code is to compare with textual data files produced by the Nortek software. In March 2016, an `oce` user shared a dataset of the `c3` variety, and this permitted detailed comparison between the text file and the values inferred by `read.adv.nortek`. The test suggested agreement (to within the resolution printed in the text file) for velocity (`v` in the `data` slot), signal amplitude (`a`), correlation (`q`), pressure (`p`), the three components of IMU delta angle (`IMUdeltaAngleX` etc), and all components of the rotation matrix (`IMUrotation`). However, the delta velocity signals did not match, with `IMUdeltaVelocityX` disagreeing in the second decimal place, `IMUdeltaVelocityY` component disagreeing in the first, and `IMUdeltaVelocityZ` being out by a factor of about 10. This is github issue 893 (https://github.com/dankelley/oce/issues/893).

- Variety c3 (signalled by byte 5 of a sequence being 0xc3) provides information on what Nortek calls DeltaAngle, DeltaVelocity and Orientation Matrix. (Apart from the orientation matrix, Nortek provides no documentation on what these quantities mean.) In the object returned by read.adv.nortek, these are stored in the data slot as IMUdeltaAngleX, IMUdeltaAngleY, IMUdeltaAngleZ, IMUdeltaVelocityX, IMUdeltaVelocityY, IMUdeltaVelocityZ, and IMUrotation, all vectors except the last, which is a 3D array. In addition to these, IMUtimestamp is a time-stamp, which is not defined in the Nortek documents but seems, from IMU documents [5], to be defined based on a clock that ticks once per 16 microseconds. Caution may be required in dealing with this timestamp, since it seemed sensible in one test case (variety d3) but kept reseting to zero in another (variety c3). The lack of Nortek documentation on most of these quantities is a roadblock to implementing oce functions dealing with IMU-enabled datasets
- Variety cc (signalled by byte 5 of a sequence being 0xcc) provides information on acceleration, angular rotation rate, magnetic vector and orientation matrix. Each is a timeseries. Acceleration is stored in the data slot as IMUaccelX, IMUaccelY, IMUaccelz. The angular rotation components are IMUngrtX, IMUngrtY and IMUngrtz. The magnetic data are in IMUmagrtx, IMUmagrty and IMUmagrtz. Finally, IMUmatrix is a rotation matrix made up from elements named M11, M12, etc in the Nortek documentation. In addition to all of these, IMUtime stores time in seconds (with an origin whose definition is not stated in [1B]).
- Variety d2 (signalled by byte 5 being 0xd2) provides information on gyro-stabilized acceleration, angular rate and magnetometer vectors. The data stored MUaccelX, IMUangrtX, IMUmagrtX, with similar for Y and Z. Again, time is in IMUtime. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.
- Variety d3 (signalled by byte 5 being 0xd3) provides information on DeltaAngle, DeltaVelocity and magnetometer vectors, stored in IMUdeltaAngleX, IMUdeltaVelocityX, and IMUdeltaMagVectorX, with similar for Y and Z. Again, time is in IMUtime. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.

Author(s)

Dan Kelley

References

1A. Nortek AS. System Integrator Guide (paradopp family of products). June 2008. (Doc No: PSI00-0101-0608). (Users may find it helpful to also examine newer versions of the guide.)

1B. Nortek AS. System Integrator Manual. Dec 2014. (system-integrator-manual_Dec2014_jan.pdf)

1C. Nortek AS. System Integrator Manual. March 2016. (system-integrator-manual_Mar2016.pdf)

2. SonTek/YSI ADVField/Hydra Acoustic Doppler Velocimeter (Field) Technical Documentation (Sept 1, 2001).

3. Appendix 2.2.3 of the Sontek ADV operation Manual, Firmware Version 4.0 (Oct 1997).

4. Nortek Knowledge Center http://www.nortekusa.com/en/knowledge-center

5. A document describing an IMU unit that seems to be close to the one named in [1B,C] as being an adjunct to Nortek Vector systems is at http://files.microstrain.com/3DM-GX3-35-Data-Communications-Protocol.

See Also

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv-class, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

Examples

```
## Not run:
library(oce)
# A nortek Vector file
d <- read.oce("/data/archive/sleiwex/2008/moorings/m05/adv/nortek_1943/raw/adv_nortek_1943.vec",
              from=as.POSIXct("2008-06-26 00:00:00", tz="UTC"),
              to=as.POSIXct("2008-06-26 00:00:10", tz="UTC"))
plot(d, which=c(1:3,15))

## End(Not run)
```

---

read.adv.sontek.serial   *Read an ADV data file*

Description

Read an ADV data file, producing an object of type adv. This function works by transferring control to a more specialized function, e.g. read.adp.nortek and read.adp.sontek, and in many cases users will find it preferable to either use these or the several even more specialized functions, if the file type is known.

Usage

```
read.adv.sontek.serial(file, from = 1, to, by = 1,
  tz = getOption("oceTz"), longitude = NA, latitude = NA, start = NULL,
  deltat = NULL, debug = getOption("oceDebug"), monitor = FALSE,
  processingLog = NULL)
```

Arguments

file	a connection or a character string giving the name of the file to load. It is also possible to give file as a vector of filenames, to handle the case of data split into files by a data logger. In the multi-file case, header must be FALSE, start must be a vector of times, and deltat must be provided.
from	index number of the first profile to be read, or the time of that profile, as created with as.POSIXct (hint: use tz="UTC"). This argument is ignored if header==FALSE. See "Examples".
to	indication of the last profile to read, in a format matching that of from. This is ignored if header==FALSE.

by	an indication of the stride length to use while walking through the file. This is ignored if `header==FALSE`. Otherwise, if this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read. This may not make much sense, if the data are not equi-spaced in time. If `by` is a string representing a time interval, in colon-separated format, then this interval is divided by the sampling interval, to get the stride length. *BUG:* `by` only partially works; see the "Bugs" section below.
tz	character string indicating time zone to be assumed in the data.
longitude	optional signed number indicating the longitude in degrees East.
latitude	optional signed number indicating the latitude in degrees North.
start	the time of the first sample, typically created with `as.POSIXct`. This may be a vector of times, if `filename` is a vector of file names.
deltat	the time between samples.
debug	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies. For example, `read.adv.nortek()` calls `read.header.nortek()`, so that `read.adv.nortek(...,debug=2)` provides information about not just the main body of the data file, but also the details of the header.
monitor	boolean, set to `TRUE` to provide an indication of every data burst read.
processingLog	if provided, the action item to be stored in the log. This parameter is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

Details

Files *without* headers may be created in experiments in which a data logger was set up to monitor the serial data stream from an instrument. The lack of header information places a burden on the user, who must supply such basic information as the times of observations, the instrument orientation, the instrument coordinate system, etc. Example 3 below shows how to deal with such files. Three things should be noted.

1. The user must choose the appropriate `read.adv` variant corresponding to the instrument in question. (This is necessary because `oceMagic`, which is used by the generic `read.oce` routine, cannot determine the type of instrument by examining a file that lacks a header.)
2. The call to the `read` function must include a start time (`start`) and the number of seconds between data (`deltat`), again, because the instrument data stream may lack those things when the device is set to a serial mode. Also, of course, it is necessary to set `header=FALSE` in the function call.
3. Once the file has been read in, the user will be obliged to specify other information, for the object to be well-formed. For example, the `read` function will have no way of knowing the instrument orientation, the coordinate system being used, the transformation matrix to go from `"beam"` to `"xyz"` coordinates, or the instrument heading, pitch, and roll, to go from `"xyz"` coordinates to `"enu"` coordinates. Such things are illustrated in example 3 below.

In ADV data files, velocities are coded to signed 2-byte integers, with a scale factor being used to convert to velocity in metres per second. These two facts control the maximum recordable velocity and the velocity resolution, values that may be retrieved for an ADV object name `d` with `d[["velocityMaximum"]]` and `d[["velocityResolution"]]`.

Value

An object of `adv-class` that contains measurements made with an ADV device.

The `metadata` contains information as given in the following table. The "Nortek name" is the name used in the Nortek System Integrator Guide [reference 1] and the "Sontek name" is the name used in the relevant Sontek documentation. References are given in square brackets.

`metadata` **name**	**Nortek name**	**Sontek name**	**Meaning**
`manufacturer`	-	-	Either `"nortek"` or `"sontek"`
`instrumentType`	-	-	Either `"vector"` or `"adv"`
`filename`	-	-	Name of data file(s)
`latitude`	-	-	Latitude of mooring (if applicable)
`longitude`	-	-	Longitude of mooring (if applicable)
`numberOfSamples`	-	-	Number of data samples in file
`numberOfBeams`	NBeams [1 p18]	-	Number of beams (always 3)
`numberOfBeamSequencesPerBurst`	NPings	-	number of beam sequences per burst
`measurementInterval`	MeasInterval [1 p31]	-	
`samplingRate`	512/(AvgInterval) [1 p30; 4]	-	

The `data` list contains items with names corresponding to `adp` objects, with an exception for Nortek data. Nortek instruments report some things at a time interval that is longer than the velocity sampling, and these are stored in `data` as `timeSlow`, `headingSlow`, `pitchSlow`, `rollSlow`, and `temperatureSlow`; if burst sampling was used, there will also be items `recordsBurst` and `timeBurst`.

The `processingLog` is in the standard format.

Nortek files

Sampling-rate and similar issues

The data format is inferred from the System Integrator Guide [1A] and System Integrator Manual [1B]. These document lacks clarity in spots, and so `read.adv.nortek` contains some assumptions that are noted here, so that users will be aware of possible problems.

A prominent example is the specification of the sampling rate, stored in `metadata$sampingRate` in the return value. Repeated examination of the System Integrator Guide [1] failed to indicate where this value is stored in the various headers contained in Vector datasets. After some experimentation with a few data files, `read.adv.nortek` was set up to calculate `metadata$samplingRate` as `512/AvgInterval` where `AvgInterval` is a part of the "User Configuration" header [1 p30], where the explanation is "average interval in seconds"). This formula was developed through trial and error, but it was confirmed later on the Nortek discussion group, and it should appear in upcoming versions of [1].

Another basic issue is the determination of whether an instrument had recorded in continuous mode or burst mode. One might infer that `TimCtrlReg` in the "User Configuration" header [1 p30] determines this, in bits 1 and 2. However, this was the case in test files available to the author. For this reason, `read.adv.nortek` infers the mode by reverse engineering of data files of known configuration. The present version of `read.adv.nortek` determines the sampling mode from the "`NRecords`" item of the "Vector Velocity Data" header, which seems to be 0 for data collected continuously, and non-zero for data collected in bursts.

Taking these things together, we come upon the issue of how to infer sampling times for Nortek instruments. There do not seem to be definitive documents on this, and so `read.adv.nortek` is based partly on information (of unknown quality) found on Nortek discusson boards. The present version of `read.adv.nortek` infers the times of velocity observations differently, depending on whether the instrument was set to record in burst mode or continuous mode. For burst mode, times stated in the burst headers are used, but for continous mode, times stated in the “vector system data” are used. On the advice found on a Nortek discussion board, the burst-mode times are offset by 2 seconds to allow for the instrument warm-up period.

Handling IMU (inertial measurement unit) data

Starting in March 2016, `read.adv.nortek` has offered some support for handling IMU (inertial measurement unit) data incorporated into Nortek binary files. This is not described in the Nortek document named “System Integrator Guide” (2008 [1A]) but it appeared in “System Integrator Manual” (2014 [1B]; 2016 [1C]). Confusingly, 1B described 3 varieties of data, whereas 1C does not describe any of these, but describes instead a fourth variety. As of March 2016, `read.adv.nortek` handles all 4 varieties, because files in the various schemes appear to exist. In `oce`, the varieties are named after the byte code that flags them. (Variety `c3` is the one described in [1C]; the others were described in [1B].) The variety is stored in the `metadata` slot of the returned object as a string named `IMUtype`.

For each variety, the reader is cautioned that strong tests have not been performed on the code. One way to test the code is to compare with textual data files produced by the Nortek software. In March 2016, an `oce` user shared a dataset of the `c3` variety, and this permitted detailed comparison between the text file and the values inferred by `read.adv.nortek`. The test suggested agreement (to within the resolution printed in the text file) for velocity (`v` in the `data` slot), signal amplitude (`a`), correlation (`q`), pressure (`p`), the three components of IMU delta angle (`IMUdeltaAngleX` etc), and all components of the rotation matrix (`IMUrotation`). However, the delta velocity signals did not match, with `IMUdeltaVelocityX` disagreeing in the second decimal place, `IMUdeltaVelocityY` component disagreeing in the first, and `IMUdeltaVelocityZ` being out by a factor of about 10. This is github issue 893 (https://github.com/dankelley/oce/issues/893).

- Variety `c3` (signalled by byte 5 of a sequence being `0xc3`) provides information on what Nortek calls DeltaAngle, DeltaVelocity and Orientation Matrix. (Apart from the orientation matrix, Nortek provides no documentation on what these quantities mean.) In the object returned by `read.adv.nortek`, these are stored in the `data` slot as `IMUdeltaAngleX`, `IMUdeltaAngleY`, `IMUdeltaAngleZ`, `IMUdeltaVelocityX`, `IMUdeltaVelocityY`, `IMUdeltaVelocityZ`, and `IMUrotation`, all vectors except the last, which is a 3D array. In addition to these, `IMUtimestamp` is a timestamp, which is not defined in the Nortek documents but seems, from IMU documents [5], to be defined based on a clock that ticks once per 16 microseconds. Caution may be required in dealing with this timestamp, since it seemed sensible in one test case (variety `d3`) but kept reseting to zero in another (variety `c3`). The lack of Nortek documentation on most of these quantities is a roadblock to implementing `oce` functions dealing with IMU-enabled datasets
- Variety `cc` (signalled by byte 5 of a sequence being `0xcc`) provides information on acceleration, angular rotation rate, magnetic vector and orientation matrix. Each is a timeseries. Acceleration is stored in the `data` slot as `IMUaccelX`, `IMUaccelY`, `IMUaccelz`. The angular rotation components are `IMUngrtX`, `IMUngrtY` and `IMUngrtz`. The magnetic data are in `IMUmagrtx`, `IMUmagrty` and `IMUmagrtz`. Finally, `IMUmatrix` is a rotation matrix made up from elements named `M11`, `M12`, etc in the Nortek documentation. In addition to all of these, `IMUtime` stores time in seconds (with an origin whose definition is not stated in [1B]).

- Variety d2 (signalled by byte 5 being `0xd2`) provides information on gyro-stabilized acceleration, angular rate and magnetometer vectors. The data stored `MUaccelX`, `IMUangrtX`, `IMUmagrtX`, with similar for `Y` and `Z`. Again, time is in `IMUtime`. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.
- Variety d3 (signalled by byte 5 being `0xd3`) provides information on DeltaAngle, DeltaVelocity and magnetometer vectors, stored in `IMUdeltaAngleX`, `IMUdeltaVelocityX`, and `IMUdeltaMagVectorX`, with similar for `Y` and `Z`. Again, time is in `IMUtime`. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.

Author(s)

Dan Kelley

References

1A. Nortek AS. System Integrator Guide (paradopp family of products). June 2008. (Doc No: PSI00-0101-0608). (Users may find it helpful to also examine newer versions of the guide.)

1B. Nortek AS. System Integrator Manual. Dec 2014. (`system-integrator-manual_Dec2014_jan.pdf`)

1C. Nortek AS. System Integrator Manual. March 2016. (`system-integrator-manual_Mar2016.pdf`)

2. SonTek/YSI ADVField/Hydra Acoustic Doppler Velocimeter (Field) Technical Documentation (Sept 1, 2001).

3. Appendix 2.2.3 of the Sontek ADV operation Manual, Firmware Version 4.0 (Oct 1997).

4. Nortek Knowledge Center `http://www.nortekusa.com/en/knowledge-center`

5. A document describing an IMU unit that seems to be close to the one named in [1B,C] as being an adjunct to Nortek Vector systems is at `http://files.microstrain.com/3DM-GX3-35-Data-Communications-Protocol.`

See Also

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOtherAdv`, `enuToOther`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnuAdv`, `xyzToEnu`

Examples

```
## Not run:
library(oce)
# A nortek Vector file
d <- read.oce("/data/archive/sleiwex/2008/moorings/m05/adv/nortek_1943/raw/adv_nortek_1943.vec",
              from=as.POSIXct("2008-06-26 00:00:00", tz="UTC"),
              to=as.POSIXct("2008-06-26 00:00:10", tz="UTC"))
plot(d, which=c(1:3,15))

## End(Not run)
```

read.adv.sontek.text *Read an ADV data file*

Description

Read an ADV data file, producing an object of type adv. This function works by transferring control to a more specialized function, e.g. read.adp.nortek and read.adp.sontek, and in many cases users will find it preferable to either use these or the several even more specialized functions, if the file type is known.

Usage

```
read.adv.sontek.text(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  originalCoordinate = "xyz", transformationMatrix, longitude = NA,
  latitude = NA, debug = getOption("oceDebug"), monitor = FALSE,
  processingLog = NULL)
```

Arguments

- `file` a connection or a character string giving the name of the file to load. It is also possible to give `file` as a vector of filenames, to handle the case of data split into files by a data logger. In the multi-file case, `header` must be `FALSE`, `start` must be a vector of times, and `deltat` must be provided.
- `from` index number of the first profile to be read, or the time of that profile, as created with `as.POSIXct` (hint: use `tz="UTC"`). This argument is ignored if `header==FALSE`. See "Examples".
- `to` indication of the last profile to read, in a format matching that of `from`. This is ignored if `header==FALSE`.
- `by` an indication of the stride length to use while walking through the file. This is ignored if `header==FALSE`. Otherwise, if this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read. This may not make much sense, if the data are not equi-spaced in time. If by is a string representing a time interval, in colon-separated format, then this interval is divided by the sampling interval, to get the stride length. *BUG:* by only partially works; see the "Bugs" section below.
- `tz` character string indicating time zone to be assumed in the data.
- `originalCoordinate` character string indicating coordinate system, one of `"beam"`, `"xyz"`, `"enu"` or `"other"`. (This is needed for the case of multiple files that were created by a data logger, because the header information is normally lost in such instances.)
- `transformationMatrix` transformation matrix to use in converting beam coordinates to xyz coordinates. This will over-ride the matrix in the file header, if there is one. An example is `rbind(c(2.710, -1.409,-1.299), c(0.071, 2.372, -2.442), c(0.344, 0.344, 0.344))`.
- `longitude` optional signed number indicating the longitude in degrees East.

`latitude`	optional signed number indicating the latitude in degrees North.
`debug`	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies. For example, `read.adv.nortek()` calls `read.header.nortek()`, so that `read.adv.nortek(...,debug=2)` provides information about not just the main body of the data file, but also the details of the header.
`monitor`	boolean, set to `TRUE` to provide an indication of every data burst read.
`processingLog`	if provided, the action item to be stored in the log. This parameter is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

Details

Files *without* headers may be created in experiments in which a data logger was set up to monitor the serial data stream from an instrument. The lack of header information places a burden on the user, who must supply such basic information as the times of observations, the instrument orientation, the instrument coordinate system, etc. Example 3 below shows how to deal with such files. Three things should be noted.

1. The user must choose the appropriate `read.adv` variant corresponding to the instrument in question. (This is necessary because `oceMagic`, which is used by the generic `read.oce` routine, cannot determine the type of instrument by examining a file that lacks a header.)
2. The call to the `read` function must include a start time (`start`) and the number of seconds between data (`deltat`), again, because the instrument data stream may lack those things when the device is set to a serial mode. Also, of course, it is necessary to set `header=FALSE` in the function call.
3. Once the file has been read in, the user will be obliged to specify other information, for the object to be well-formed. For example, the `read` function will have no way of knowing the instrument orientation, the coordinate system being used, the transformation matrix to go from `"beam"` to `"xyz"` coordinates, or the instrument heading, pitch, and roll, to go from `"xyz"` coordinates to `"enu"` coordinates. Such things are illustrated in example 3 below.

In ADV data files, velocities are coded to signed 2-byte integers, with a scale factor being used to convert to velocity in metres per second. These two facts control the maximum recordable velocity and the velocity resolution, values that may be retrieved for an ADV object name `d` with `d[["velocityMaximum"]]` and `d[["velocityResolution"]]`.

Value

An object of `adv-class` that contains measurements made with an ADV device.

The `metadata` contains information as given in the following table. The “Nortek name” is the name used in the Nortek System Integrator Guide [reference 1] and the “Sontek name” is the name used in the relevant Sontek documentation. References are given in square brackets.

`metadata` **name**	**Nortek name**	**Sontek name**	**Meaning**
`manufacturer`	-	-	Either `"nortek"` or `"sontek"`
`instrumentType`	-	-	Either `"vector"` or `"adv"`
`filename`	-	-	Name of data file(s)

`latitude`	-	-	Latitude of mooring (if applicable)
`longitude`	-	-	Longitude of mooring (if applicable)
`numberOfSamples`	-	-	Number of data samples in file
`numberOfBeams`	NBeams [1 p18]	-	Number of beams (always 3)
`numberOfBeamSequencesPerBurst`	NPings	-	number of beam sequences per burst
`measurementInterval`	MeasInterval [1 p31]	-	
`samplingRate`	512/(AvgInterval) [1 p30; 4]	-	

The `data` list contains items with names corresponding to `adp` objects, with an exception for Nortek data. Nortek instruments report some things at a time interval that is longer than the velocity sampling, and these are stored in `data` as `timeSlow`, `headingSlow`, `pitchSlow`, `rollSlow`, and `temperatureSlow`; if burst sampling was used, there will also be items `recordsBurst` and `timeBurst`.

The `processingLog` is in the standard format.

Nortek files

Sampling-rate and similar issues

The data format is inferred from the System Integrator Guide [1A] and System Integrator Manual [1B]. These document lacks clarity in spots, and so `read.adv.nortek` contains some assumptions that are noted here, so that users will be aware of possible problems.

A prominent example is the specification of the sampling rate, stored in `metadata$sampingRate` in the return value. Repeated examination of the System Integrator Guide [1] failed to indicate where this value is stored in the various headers contained in Vector datasets. After some experimentation with a few data files, `read.adv.nortek` was set up to calculate `metadata$samplingRate` as `512/AvgInterval` where `AvgInterval` is a part of the "User Configuration" header [1 p30], where the explanation is "average interval in seconds"). This formula was developed through trial and error, but it was confirmed later on the Nortek discussion group, and it should appear in upcoming versions of [1].

Another basic issue is the determination of whether an instrument had recorded in continuous mode or burst mode. One might infer that `TimCtrlReg` in the "User Configuration" header [1 p30] determines this, in bits 1 and 2. However, this was the case in test files available to the author. For this reason, `read.adv.nortek` infers the mode by reverse engineering of data files of known configuration. The present version of `read.adv.nortek` determines the sampling mode from the "`NRecords`" item of the "Vector Velocity Data" header, which seems to be 0 for data collected continuously, and non-zero for data collected in bursts.

Taking these things together, we come upon the issue of how to infer sampling times for Nortek instruments. There do not seem to be definitive documents on this, and so `read.adv.nortek` is based partly on information (of unknown quality) found on Nortek discusson boards. The present version of `read.adv.nortek` infers the times of velocity observations differently, depending on whether the instrument was set to record in burst mode or continuous mode. For burst mode, times stated in the burst headers are used, but for continous mode, times stated in the "vector system data" are used. On the advice found on a Nortek discussion board, the burst-mode times are offset by 2 seconds to allow for the instrument warm-up period.

Handling IMU (inertial measurement unit) data

Starting in March 2016, `read.adv.nortek` has offered some support for handling IMU (inertial measurement unit) data incorporated into Nortek binary files. This is not described in the Nortek document named “System Integrator Guide” (2008 [1A]) but it appeared in “System Integrator Manual” (2014 [1B]; 2016 [1C]). Confusingly, 1B described 3 varieties of data, whereas 1C does not describe any of these, but describes instead a fourth variety. As of March 2016, `read.adv.nortek` handles all 4 varieties, because files in the various schemes appear to exist. In `oce`, the varieties are named after the byte code that flags them. (Variety `c3` is the one described in [1C]; the others were described in [1B].) The variety is stored in the `metadata` slot of the returned object as a string named `IMUtype`.

For each variety, the reader is cautioned that strong tests have not been performed on the code. One way to test the code is to compare with textual data files produced by the Nortek software. In March 2016, an `oce` user shared a dataset of the `c3` variety, and this permitted detailed comparison between the text file and the values inferred by `read.adv.nortek`. The test suggested agreement (to within the resolution printed in the text file) for velocity (`v` in the `data` slot), signal amplitude (`a`), correlation (`q`), pressure (`p`), the three components of IMU delta angle (`IMUdeltaAngleX` etc), and all components of the rotation matrix (`IMUrotation`). However, the delta velocity signals did not match, with `IMUdeltaVelocityX` disagreeing in the second decimal place, `IMUdeltaVelocityY` component disagreeing in the first, and `IMUdeltaVelocityZ` being out by a factor of about 10. This is github issue 893 (https://github.com/dankelley/oce/issues/893).

- Variety `c3` (signalled by byte 5 of a sequence being `0xc3`) provides information on what Nortek calls DeltaAngle, DeltaVelocity and Orientation Matrix. (Apart from the orientation matrix, Nortek provides no documentation on what these quantities mean.) In the object returned by `read.adv.nortek`, these are stored in the `data` slot as `IMUdeltaAngleX`, `IMUdeltaAngleY`, `IMUdeltaAngleZ`, `IMUdeltaVelocityX`, `IMUdeltaVelocityY`, `IMUdeltaVelocityZ`, and `IMUrotation`, all vectors except the last, which is a 3D array. In addition to these, `IMUtimestamp` is a timestamp, which is not defined in the Nortek documents but seems, from IMU documents [5], to be defined based on a clock that ticks once per 16 microseconds. Caution may be required in dealing with this timestamp, since it seemed sensible in one test case (variety `d3`) but kept reseting to zero in another (variety `c3`). The lack of Nortek documentation on most of these quantities is a roadblock to implementing `oce` functions dealing with IMU-enabled datasets
- Variety `cc` (signalled by byte 5 of a sequence being `0xcc`) provides information on acceleration, angular rotation rate, magnetic vector and orientation matrix. Each is a timeseries. Acceleration is stored in the `data` slot as `IMUaccelX`, `IMUaccelY`, `IMUaccelz`. The angular rotation components are `IMUngrtX`, `IMUngrtY` and `IMUngrtz`. The magnetic data are in `IMUmagrtx`, `IMUmagrty` and `IMUmagrtz`. Finally, `IMUmatrix` is a rotation matrix made up from elements named `M11`, `M12`, etc in the Nortek documentation. In addition to all of these, `IMUtime` stores time in seconds (with an origin whose definition is not stated in [1B]).
- Variety `d2` (signalled by byte 5 being `0xd2`) provides information on gyro-stabilized acceleration, angular rate and magnetometer vectors. The data stored `MUaccelX`, `IMUangrtX`, `IMUmagrtX`, with similar for `Y` and `Z`. Again, time is in `IMUtime`. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.
- Variety `d3` (signalled by byte 5 being `0xd3`) provides information on DeltaAngle, DeltaVelocity and magnetometer vectors, stored in `IMUdeltaAngleX`, `IMUdeltaVelocityX`, and `IMUdeltaMagVectorX`, with similar for `Y` and `Z`. Again, time is in `IMUtime`. This data type has not been tested as of mid-March 2016, because the developers do not have a test file with which to test.

Note on file name

The `file` argument does not actually name a file. It names a basename for a file. The actual file names are created by appending suffix `.hd1` for one file and `.ts1` for another.

Author(s)

Dan Kelley

References

1A. Nortek AS. System Integrator Guide (paradopp family of products). June 2008. (Doc No: PSI00-0101-0608). (Users may find it helpful to also examine newer versions of the guide.)

1B. Nortek AS. System Integrator Manual. Dec 2014. (`system-integrator-manual_Dec2014_jan.pdf`)

1C. Nortek AS. System Integrator Manual. March 2016. (`system-integrator-manual_Mar2016.pdf`)

2. SonTek/YSI ADVField/Hydra Acoustic Doppler Velocimeter (Field) Technical Documentation (Sept 1, 2001).

3. Appendix 2.2.3 of the Sontek ADV operation Manual, Firmware Version 4.0 (Oct 1997).

4. Nortek Knowledge Center `http://www.nortekusa.com/en/knowledge-center`

5. A document describing an IMU unit that seems to be close to the one named in [1B,C] as being an adjunct to Nortek Vector systems is at `http://files.microstrain.com/3DM-GX3-35-Data-Communications-Protocol.`

See Also

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOtherAdv`, `enuToOther`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnuAdv`, `xyzToEnu`

Examples

```
## Not run:
library(oce)
# A nortek Vector file
d <- read.oce("/data/archive/sleiwex/2008/moorings/m05/adv/nortek_1943/raw/adv_nortek_1943.vec",
              from=as.POSIXct("2008-06-26 00:00:00", tz="UTC"),
              to=as.POSIXct("2008-06-26 00:00:10", tz="UTC"))
plot(d, which=c(1:3,15))

## End(Not run)
```

read.amsr *Read an amsr File*

Description

Read a compressed amsr file, generating an object that inherits from amsr-class. Note that only compressed files are read in this version.

Usage

```
read.amsr(file, debug = getOption("oceDebug"))
```

Arguments

file	String indicating the name of a compressed file. See "File sources".
debug	A debugging flag, integer.

File sources

AMSR files are provided at the FTP site `ftp://ftp.ssmi.com/amsr2/bmaps_v07.2/` and login as "guest", enter a year-based directory (e.g. `y2016` for the year 2016), then enter a month-based directory (e.g. `m08` for August, the 8th month), and then download a file for the present date, e.g. `f34_20160803v7.2.gz` for August 3rd, 2016. Do not uncompress this file, since `read.amsr` can only read uncompressed files. If `read.amsr` reports an error on the number of chunks, try downloading a similarly-named file (e.g. in the present example, `read.amsr("f34_20160803v7.2_d3d.gz")` will report an error about inability to read a 6-chunk file, but `read.amsr("f34_20160803v7.2.gz")` will work properly.

Author(s)

Dan Kelley and Chantelle Layton

See Also

plot,amsr-method for an example.

Other things related to amsr data: [[<-,amsr-method, amsr-class, composite,amsr-method, download.amsr, plot,amsr-method, subset,amsr-method, summary,amsr-method `read.aquadopp` *Read a Nortek Aquadopp File*

Description

The R code is based on information in the Nortek System Integrator Guide (2008) and on postings on the Nortek "knowledge center" discussion board. One might assume that the latter is less authoritative than the former. For example, the inference of cell size follows advice found at http://www.nortekusa.com/en/knowledge-center/forum/hr-profilers/736804717 (downloaded June 2012), which contains a typo in an early posting that is corrected later on.

Usage

```
read.aquadopp(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  longitude = NA, latitude = NA, orientation, distance, monitor = FALSE,
  despike = FALSE, processingLog, debug = getOption("oceDebug"), ...)
```

Arguments

`file`	a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
`from`	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
`to`	an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.
`by`	an optional indication of the stride length to use while walking through the file. If this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read, e.g. the default `by=1` means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see "Memory considerations".)
`tz`	character string indicating time zone to be assumed in the data.
`longitude`	optional signed number indicating the longitude in degrees East.
`latitude`	optional signed number indicating the latitude in degrees North.
`orientation`	Optional character string specifying the orientation of the sensor, provided for those cases in which it cannot be inferred from the data file. The valid choices are `"upward"`, `"downward"`, and `"sideward"`.
`distance`	Optional vector holding the distances of bin centres from the sensor. This argument is ignored except for Nortek profilers, and need not be given if the function determines the distances correctly from the data. The problem is that the

	distance is poorly documented in the Nortek System Integrator Guide (2008 edition, page 31), so the function must rely on word-of-mouth formulae that do not work in all cases.
`monitor`	boolean, set to `TRUE` to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with `txtProgressBar`.
`despike`	if `TRUE`, `despike` will be used to clean anomalous spikes in heading, etc.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	additional arguments, passed to called routines.

Value

An adp object, i.e. one inheriting from `adp-class`.

Author(s)

Dan Kelley

References

1. Information on Nortek profilers (including the System Integrator Guide, which explains the data format byte-by-byte) is available at `http://www.nortekusa.com/`. (One must join the site to see the manuals.)

2. The Nortek Knowledge Center `http://www.nortekusa.com/en/knowledge-center` may be of help if problems arise in dealing with data from Nortek instruments.

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

`read.aquadoppHR` *Read Nortek Aquadopp-HR File*

Description

The R code is based on information in the Nortek System Integrator Guide (2008) and on postings on the Nortek "knowledge center" discussion board. One might assume that the latter is less authoritative than the former. For example, the inference of cell size follows advice found at `http://www.nortekusa.com/en/knowledge-center/forum/hr-profilers/736804717` (downloaded June 2012)), which contains a typo in an early posting that is corrected later on.

Usage

```
read.aquadoppHR(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  longitude = NA, latitude = NA, orientation = orientation, distance,
  monitor = FALSE, despike = FALSE, processingLog,
  debug = getOption("oceDebug"), ...)
```

Arguments

`file`	a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
`from`	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
`to`	an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.
`by`	an optional indication of the stride length to use while walking through the file. If this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read, e.g. the default `by=1` means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see "Memory considerations".)
`tz`	character string indicating time zone to be assumed in the data.
`longitude`	optional signed number indicating the longitude in degrees East.
`latitude`	optional signed number indicating the latitude in degrees North.
`orientation`	Optional character string specifying the orientation of the sensor, provided for those cases in which it cannot be inferred from the data file. The valid choices are `"upward"`, `"downward"`, and `"sideward"`.
`distance`	Optional vector holding the distances of bin centres from the sensor. This argument is ignored except for Nortek profilers, and need not be given if the function determines the distances correctly from the data. The problem is that the distance is poorly documented in the Nortek System Integrator Guide (2008 edition, page 31), so the function must rely on word-of-mouth formulae that do not work in all cases.
`monitor`	boolean, set to `TRUE` to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with `txtProgressBar`.
`despike`	if `TRUE`, `despike` will be used to clean anomalous spikes in heading, etc.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	additional arguments, passed to called routines.

Value

An adp object, i.e. one inheriting from adp-class.

Author(s)

Dan Kelley

References

1. Information on Nortek profilers (including the System Integrator Guide, which explains the data format byte-by-byte) is available at http://www.nortekusa.com/. (One must join the site to see the manuals.)

2. The Nortek Knowledge Center http://www.nortekusa.com/en/knowledge-center may be of help if problems arise in dealing with data from Nortek instruments.

See Also

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

---

read.aquadoppProfiler *Read a Nortek Aquadopp-Profiler File*

Description

The R code is based on information in the Nortek System Integrator Guide (2008) and on postings on the Nortek "knowledge center" discussion board. One might assume that the latter is less authoritative than the former. For example, the inference of cell size follows advice found at http://www.nortekusa.com/en/knowledge-center/forum/hr-profilers/736804717 (downloaded June 2012)), which contains a typo in an early posting that is corrected later on.

Usage

```
read.aquadoppProfiler(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  longitude = NA, latitude = NA, orientation, distance, monitor = FALSE,
  despike = FALSE, processingLog, debug = getOption("oceDebug"), ...)
```

Arguments

- `file` — a connection or a character string giving the name of the file to load. (For `read.adp.sontek.serial`, this is generally a list of files, which will be concatenated.)
- `from` — indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the `tz` argument. If `from` is not supplied, it defaults to 1.
- `to` — an optional indication of the last profile to read, in a format as described for `from`. As a special case, `to=0` means to read the file to the end. If `to` is not supplied, then it defaults to 0.
- `by` — an optional indication of the stride length to use while walking through the file. If this is an integer, then `by-1` profiles are skipped between each pair of profiles that is read, e.g. the default `by=1` means to read all the data. (For RDI files *only*, there are some extra features to avoid running out of memory; see "Memory considerations".)
- `tz` — character string indicating time zone to be assumed in the data.
- `longitude` — optional signed number indicating the longitude in degrees East.
- `latitude` — optional signed number indicating the latitude in degrees North.
- `orientation` — Optional character string specifying the orientation of the sensor, provided for those cases in which it cannot be inferred from the data file. The valid choices are `"upward"`, `"downward"`, and `"sideward"`.
- `distance` — Optional vector holding the distances of bin centres from the sensor. This argument is ignored except for Nortek profilers, and need not be given if the function determines the distances correctly from the data. The problem is that the distance is poorly documented in the Nortek System Integrator Guide (2008 edition, page 31), so the function must rely on word-of-mouth formulae that do not work in all cases.
- `monitor` — boolean, set to `TRUE` to provide an indication of progress in reading the file, either by printing a dot for each profile or by writing a textual progress bar with `txtProgressBar`.
- `despike` — if `TRUE`, `despike` will be used to clean anomalous spikes in heading, etc.
- `processingLog` — if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
- `debug` — a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
- `...` — additional arguments, passed to called routines.

Value

An adp object, i.e. one inheriting from `adp-class`.

Author(s)

Dan Kelley

References

1. Information on Nortek profilers (including the System Integrator Guide, which explains the data format byte-by-byte) is available at http://www.nortekusa.com/. (One must join the site to see the manuals.)

2. The Nortek Knowledge Center http://www.nortekusa.com/en/knowledge-center may be of help if problems arise in dealing with data from Nortek instruments.

See Also

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

---

read.argo — *Read an Argo Data File*

Description

read.argo is used to read an Argo file, producing an object of type argo. The file must be in the ARGO-style netCDF format described at in the Argo documentation [2,3].

Usage

```
read.argo(file, debug = getOption("oceDebug"), processingLog, ...)
```

Arguments

file	a character string giving the name of the file to load.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
...	additional arguments, passed to called routines.

Details

Metadata items such as time, longitude and latitude are inferred from the data file in a straightforward way, using ncvar_get and data-variable names as listed in the Argo documentation [2,3]. The items listed in section 2.2.3 of [3] is read from the file and stored in the metadata slot, with the exception of longitude and latitude, which are stored in the data slot.

String data that contain trailing blanks in the argo NetCDF are trimmed using trimString. One-dimensional matrices are converted to vectors using as.vector. Items listed in section 2.2.3 of [3] are meant to be present in all files, but tests showed that this is not the case, and so read.argo sets such items to NULL before saving them in returned object.

Items are translated from upper-case Argo names to oce names using argoNames2oceNames.

It is assumed that the profile data are as listed in the NetCDF variable called STATION_PARAMETERS. Each item can have variants, as described in Sections 2.3.4 of [3]. For example, if "PRES" is found in STATION_PARAMETERS, then PRES (pressure) data are sought in the file, along with PRES_QC, PRES_ADJUSTED, PRES_ADJUSTED_QC, and PRES_ERROR. The same pattern works for other profile data. The variables are stored with different names within the resultant argo-class object, to match with oce conventions. Thus, PRES gets renamed pressure, while PRES_ADJUSTED gets renamed pressureAdjusted, and PRES_ERROR gets renamed pressureError; all of these are stored in the data slot. Meanwhile, the quality-control flags PRES_QC and PRES_ADJUSTED_QC are stored as pressure and pressureAdjusted in the metadata$flags slot.

Value

An object of argo-class.

Data sources

Argo data are made available at several websites. A bit of detective work can be required to track down the data.

Some servers provide data for floats that surfaced in a given ocean on a given day, the anonymous FTP server ftp://usgodae.org/pub/outgoing/argo/geo/ being an example.

Other servers provide data on a per-float basis. A complicating factor is that these data tend to be categorized by "dac" (data archiving centre), which makes it difficult to find a particular float. For example, http://www.usgodae.org/ftp/outgoing/argo/ is the top level of a such a repository. If the ID of a float is known but not the "dac", then a first step is to download the text file http://www.usgodae.org/ftp/outgoing/argo/ar_index_global_meta.txt and search for the ID. The first few lines of that file are header, and after that the format is simple, with columns separated by slash (/). The dac is in the first such column and the float ID in the second. A simple search will reveal the dac. For example data(argo) is based on float 6900388, and the line containing that token is bodc/6900388/6900388_meta.nc,846,BO,20120225005617, from which the dac is seen to be the British Oceanographic Data Centre (bodc). Armed with that information, visit http://www.usgodae.org/ftp/outgoing/argo/dac/bodc/6900388 and see a directory called 'profiles' that contains a NetCDF file for each profile the float made. These can be read with read.argo. It is also possible, and probably more common, to read a NetCDF file containing all the profiles together and for that purpose the file http://www.usgodae.org/ftp/outgoing/argo/dac/bodc/6900388/6900388_prof.nc should be downloaded and provided as the file argument to read.argo. This can be automated as in Example 2, although readers are cautioned that URL structures tend to change over time.

Similar steps can be followed on other servers.

Author(s)

Dan Kelley

References

1. http://www.argo.ucsd.edu/

2. Argo User's Manual Version 3.2, Dec 29th, 2015, available at https://archimer.ifremer.fr/doc/00187/29825/40575.pdf (but note that this is a draft; newer versions may have replaced this by now).

3. User's Manual (ar-um-02-01) 13 July 2010, available at http://www.argodatamgt.org/content/download/4729/34634/file/argo-dm-user-manual-version-2.3.pdf, which is the main document describing argo data.

See Also

The documentation for argo-class explains the structure of argo objects, and also outlines the other functions dealing with them.

Other things related to argo data: [[,argo-method, [[<-,argo-method, argo-class, argoGrid, argoNames2oceNames, argo, as.argo, handleFlags,argo-method, plot,argo-method, subset,argo-method, summary,argo-method

Examples

```
## Not run:
## Example 1: read from a local file
library(oce)
d <- read.argo("/data/OAR/6900388_prof.nc")
summary(d)
plot(d)

## Example 2: construct URL for download (brittle)
id <- "6900388"
url <- "http://www.usgodae.org/ftp/outgoing/argo"
if (!length(list.files(pattern="argo_index.txt")))
    download.file(paste(url, "ar_index_global_meta.txt", sep="/"), "argo_index.txt")
index <- readLines("argo_index.txt")
line <- grep(id, index)
if (0 == length(line)) stop("id ", id, " not found")
if (1 < length(line)) stop("id ", id, " found multiple times")
dac <- strsplit(index[line], "/")[[1]][1]
profile <- paste(id, "_prof.nc", sep="")
float <- paste(url, "dac", dac, id, profile, sep="/")
download.file(float, profile)
argo <- read.argo(profile)
summary(argo)

## End(Not run)
```

read.bremen *Read a Bremen File*

Description

Read a file in Bremen format, producing an object inheriting from bremen-class.

Usage

```
read.bremen(file)
```

Arguments

file	a connection or a character string giving the name of the file to load.

Details

Velocities are assumed to be in cm/s, and are converted to m/s to follow the oce convention. Shears (which is what the variables named uz and vz are assumed to represent) are assumed to be in (cm/s)/m, although they could be in 1/s or something else; the lack of documentation is a problem here. Also, note that the assumed shears are not just first-difference estimates of velocity, given the results of a sample dataset:

```
> head(data.frame(b[["data"]]))
  pressure      u      v       uz       vz
1        0  0.092 -0.191  0.00000  0.00000
2       10  0.092 -0.191  0.02183 -0.35412
3       20  0.092 -0.191  0.03046 -0.09458
4       30  0.026 -0.246 -0.03948  0.02169
5       40 -0.003 -0.212 -0.02614  0.03111
6       50 -0.023 -0.169 -0.03791  0.01706
```

Value

An object of bremen-class.

Issues

This function may be renamed (or removed) without notice. It was created to read some data being used in a particular research project, and will be rendered useless if Bremen changes this data format.

Author(s)

Dan Kelley

See Also

Other things related to bremen data: [[,bremen-method, [[<-,bremen-method, bremen-class, plot,bremen-method, summary,bremen-method

---

read.cm *Read a CM file*

Description

Read a current-meter data file, producing an object of cm-class.

Usage

```
read.cm(file, from = 1, to, by = 1, tz = getOption("oceTz"),
  type = c("s4"), longitude = NA, latitude = NA,
  debug = getOption("oceDebug"), monitor = FALSE, processingLog, ...)
```

Arguments

file	a connection or a character string giving the name of the file to load.
from	index number of the first measurement to be read, or the time of that measurement, as created with as.POSIXct (hint: use tz="UTC").
to	indication of the last measurement to read, in a format matching that of from.
by	an indication of the stride length to use while walking through the file. If this is an integer, then by-1 measurements are skipped between each pair of profiles that is read. This may not make much sense, if the data are not equi-spaced in time. If by is a string representing a time interval, in colon-separated format, then this interval is divided by the sampling interval, to get the stride length. *BUG:* if the data are not equi-spaced, then odd results will occur.
tz	character string indicating time zone to be assumed in the data.
type	character string indicating type of file (ignored at present).
longitude	optional signed number indicating the longitude in degrees East.
latitude	optional signed number indicating the latitude in degrees North.
debug	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies.
monitor	ignored.
processingLog	if provided, the action item to be stored in the log. This parameter is typically only provided for internal calls; the default that it provides is better for normal calls by a user.
...	Optional arguments passed to plotting functions.

Details

There function has been tested on only a single file, and the data-scanning algorithm was based on visual inspection of that file. Whether it will work generally is an open question. It should be noted that the sample file had several odd characteristics, some of which are listed below.

- The file contained two columns named "Cond", which was guessed to stand for conductivity. Since only the first contained data, the second was ignored, but this may not be the case for all files.
- The unit for "Cond" was stated in the file to be "mS", which makes no sense, so the unit was assumed to be mS/cm.
- The file contained a column named "T-Temp", which is not something the author has seen in his career. It was assumed to stand for in-situ temperature.
- The file contained a column named "Depth", which is not something an instrument can measure. Presumably it was calculated from pressure (with what atmospheric offset, though?) and so pressure was inferred from it using `swPressure`.
- The file contained several columns that lacked names. These were ignored.
- The file contained several columns that seem to be derived from the actual measured data, such as "Speed", "Dir", "N-S Dist", etc. These are ignored.
- The file contained several columns that were basically a mystery to the author, e.g. "Hx", "Hy", "Vref", etc. These were ignored.

Based on such considerations, `read.cm.s4()` reads only the columns that were reasonably well-understood based on the sample file. Users who need more columns should contact the author. And a user who could produce a document explaining the data format would be especially appreciated!

Value

An object of `cm-class`. The `data` slot will contain all the data in the file, with names determined from the tokens in line 3 in that file, passed through `make.names`, except that `Vnorth` is renamed `v` (after conversion from cm/s to m/s), `Veast` is renamed `u` (after conversion from cm/s to m/s), `Cond` is renamed `conductivity`, `T.Temp` is renamed `temperature` and `Sal` is renamed `salinity`, and a new column named `time` (a POSIX time) is constructed from the information in the file header, and another named `pressure` is constructed from the column named `Depth`. At least in the single file studied in the creation of this function, there are some columns that are unnamed in line 3 of the header; these yield data items with names `X`, `X.1`, etc.

Historical note

Prior to late July, 2016, the direction of current flow was stored in the return value, but it is no longer stored, since it can be derived from the `u` and `v` values.

Author(s)

Dan Kelley

See Also

Other things related to `cm` data: `[[,cm-method`, `[[<-,cm-method`, `as.cm`, `cm-class`, `cm`, `plot,cm-method`, `subset,cm-method`, `summary,cm-method`

Examples

```
## Not run:
  library(oce)
  cm <- read.oce("cm_interocean_0811786.s4a.tab")
  summary(cm)
  plot(cm)

## End(Not run)
```

---

read.coastline   *Read a Coastline File*

Description

Read a coastline file in R, Splus, mapgen, shapefile, or openstreetmap format. The S and R formats are identical, and consist of two columns, lon and lat, with land-jump segments separated by lines with two NAs. The MapGen format is of the form

```
 # -b -16.179081 28.553943
-16.244793 28.563330
```

BUG: the 'arc/info ungenerate' format is not yet understood.

Usage

```
read.coastline(file, type = c("R", "S", "mapgen", "shapefile",
  "openstreetmap"), debug = getOption("oceDebug"), monitor = FALSE,
  processingLog)
```

Arguments

`file`	name of file containing coastline data.
`type`	type of file, one of `"R"`, `"S"`, `"mapgen"`, `"shapefile"` or `"openstreetmap"`.
`debug`	set to TRUE to print information about the header, etc.
`monitor`	print a dot for every coastline segment read (ignored except for reading "shapefile" type)
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Value

An object of `coastline-class`.

Author(s)

Dan Kelley

read.coastline.openstreetmap
Read a Coastline File in Openstreetmap Format

Description

Read coastline data stored in the openstreetmap format [1].

Usage

```
read.coastline.openstreetmap(file, lonlim = c(-180, 180), latlim = c(-90,
  90), debug = getOption("oceDebug"), monitor = FALSE, processingLog)
```

Arguments

file	name of file containing coastline data (a file ending in .shp) or a zipfile that contains such a file, with a corresponding name. The second scheme is useful for files downloaded from the NaturalEarth website [2].
lonlim	numerical vectors specifying the west and east edges (and south and north edges) of a focus window. Coastline polygons that do not intersect the defined box are skipped, which can be useful in narrowing high-resolution world-scale data to a local application.
latlim	numerical vectors specifying the west and east edges (and south and north edges) of a focus window. Coastline polygons that do not intersect the defined box are skipped, which can be useful in narrowing high-resolution world-scale data to a local application.
debug	set to TRUE to print information about the header, etc.
monitor	Logical indicating whether to print an indication of progress through the file.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Value

An object of coastline-class

Author(s)

Dan Kelley

See Also

Other things related to coastline data: [[,coastline-method, [[<-,coastline-method, as.coastline, coastline-class, coastlineBest, coastlineCut, coastlineWorld, download.coastline, plot,coastline-method, read.coastline.shapefile, subset,coastline-method, summary,coastline-method

read.coastline.shapefile
Read a Coastline File in Shapefile Format

Description

Read coastline data stored in the shapefile format [1].

Usage

```
read.coastline.shapefile(file, lonlim = c(-180, 180), latlim = c(-90, 90),
  debug = getOption("oceDebug"), monitor = FALSE, processingLog)
```

Arguments

`file`	name of file containing coastline data (a file ending in `.shp`) or a zipfile that contains such a file, with a corresponding name. The second scheme is useful for files downloaded from the NaturalEarth website [2].
`lonlim, latlim`	numerical vectors specifying the west and east edges (and south and north edges) of a focus window. Coastline polygons that do not intersect the defined box are skipped, which can be useful in narrowing high-resolution world-scale data to a local application.
`debug`	set to TRUE to print information about the header, etc.
`monitor`	Logical indicating whether to print an indication of progress through the file.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Value

An object of `coastline-class`

A hack for depth contours

The following demonstrates that this code is getting close to working with depth contours. This should be handled more internally, and a new object for depth contours should be constructed, of which coastlines could be a subset.

Author(s)

Dan Kelley

References

- 1. The “shapefile” format is described in *ESRI Shapefile Technical Description*, March 1998, available at `http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`.
- 2. The NaturalEarth website `http://www.naturalearthdata.com/downloads` provides coastline datasets in three resolutions, along with similar files lakes and rivers, for borders, etc. It is highly recommended.

See Also

Other things related to coastline data: [[,coastline-method, [[<-,coastline-method, as.coastline, coastline-class, coastlineBest, coastlineCut, coastlineWorld, download.coastline, plot,coastline-method, read.coastline.openstreetmap, subset,coastline-method, summary,coastline-method

---

read.ctd *Read a General CTD File*

Description

Read a General CTD File

Usage

```
read.ctd(file, type = NULL, columns = NULL, station = NULL, missingValue,
  monitor = FALSE, debug = getOption("oceDebug"), processingLog, ...)
```

Arguments

file	A connection or a character string giving the name of the file to load. For read.ctd.sbe() and read.ctd.woce(), this may be a wildcard (e.g. "*.cnv" or "*.csv") in which case the return value is a vector containing CTD objects created by reading the files from list.files with pattern set to the specified wildcard pattern.
type	If NULL, then the first line is studied, in order to determine the file type. If type="SBE19", then a *Seabird 19*, or similar, CTD format is assumed. If type="WOCE" then a WOCE-exchange file is assumed. If type="ITP" then an ice-tethered profiler file is assumed. If type="ODF" an ODF file is assumed. If type="ODV" an ascii-ODV file is assumed.
columns	An optional list that can be used to convert unrecognized data names to resultant variable names. This is used only by read.ctd.sbe and read.ctd.odf; see "Examples".
station	Optional character string containing an identifying name or number for the station. This can be useful if the routine cannot determine the name automatically, or if another name is preferred.
missingValue	Optional missing-value flag; data matching this value will be set to NA upon reading. If this is provided, then it overrules any missing-value flag found in the data. For Seabird (.cnv) files, there is usually no need to set missingValue, because it can be inferred from the header (typically as -9.990e-29). Set missingValue=NULL to turn off missing-value detection, even in .cnv files that contain missing-value codes in their headers. If missingValue is not specified, then an attempt is made to infer such a value from the data, by testing whether salinity and/or temperature has a minimum that is under -8 in value; this should catch common values in files, without false positives. A warning will be issued in this case, and a note inserted in the processing log of the return value.

`monitor`	Boolean, set to `TRUE` to provide an indication of progress. This is useful if `filename` is a wildcard.
`debug`	An integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed.
`processingLog`	If provided, the action item to be stored in the log. This is typically only provided for internal calls; the default that it provides is better for normal calls by a user.
`...`	additional arguments, passed to called routines.

Details

`read.ctd()` is a base function that in turn calls specialized functions, e.g. `read.ctd.odf` for the ODF data used in Fisheries and Oceans (Canada), `read.ctd.woce` for data in World Ocean Circulation Experiment format, `read.ctd.woce.other` for a variant of WOCE data, `read.ctd.itp` for ice-tethered-profiler data, or `read.ctd.sbe` for Seabird data.

Value

An object of `ctd-class`. The details of the contents depend on the source file. The `metadata` slot is particularly variable across data formats, because the meta-information provided in those formats varies widely.

Author(s)

Dan Kelley

See Also

Other things related to `ctd` data: `[[,ctd-method`, `[[<-,ctd-method`, `as.ctd`, `cnvName2oceName`, `ctd-class`, `ctdDecimate`, `ctdFindProfiles`, `ctdRaw`, `ctdTrim`, `ctd`, `handleFlags,ctd-method`, `oceNames2whpNames`, `oceUnits2whpUnits`, `plot,ctd-method`, `plotProfile`, `plotScan`, `plotTS`, `read.ctd.itp`, `read.ctd.odf`, `read.ctd.sbe`, `read.ctd.woce.other`, `read.ctd.woce`, `subset,ctd-method`, `summary,ctd-method`, `woceNames2oceNames`, `woceUnit2oceUnit`, `write.ctd`

`read.ctd.itp` *Read an ITP-type CTD File*

Description

Read an ITP-type CTD File

Usage

```
read.ctd.itp(file, columns = NULL, station = NULL, missingValue,
  monitor = FALSE, debug = getOption("oceDebug"), processingLog, ...)
```

Arguments

`file` A connection or a character string giving the name of the file to load. For `read.ctd.sbe()` and `read.ctd.woce()`, this may be a wildcard (e.g. `"*.cnv"` or `"*.csv"`) in which case the return value is a vector containing CTD objects created by reading the files from `list.files` with `pattern` set to the specified wildcard pattern.

`columns` An optional `list` that can be used to convert unrecognized data names to resultant variable names. This is used only by `read.ctd.sbe` and `read.ctd.odf`; see "Examples".

`station` Optional character string containing an identifying name or number for the station. This can be useful if the routine cannot determine the name automatically, or if another name is preferred.

`missingValue` Optional missing-value flag; data matching this value will be set to `NA` upon reading. If this is provided, then it overrules any missing-value flag found in the data. For Seabird (`.cnv`) files, there is usually no need to set `missingValue`, because it can be inferred from the header (typically as -9.990e-29). Set `missingValue=NULL` to turn off missing-value detection, even in `.cnv` files that contain missing-value codes in their headers. If `missingValue` is not specified, then an attempt is made to infer such a value from the data, by testing whether salinity and/or temperature has a minimum that is under -8 in value; this should catch common values in files, without false positives. A warning will be issued in this case, and a note inserted in the processing log of the return value.

`monitor` Boolean, set to `TRUE` to provide an indication of progress. This is useful if `filename` is a wildcard.

`debug` An integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed.

`processingLog` If provided, the action item to be stored in the log. This is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

`...` additional arguments, passed to called routines.

Details

`read.ctd.itp()` reads ice-tethered-profiler data that are stored in a format files used by WHOI servers as of 2016-2017. Lacking documentation on the format, the author constructed this function to work with some files that were on-hand. Whether the function will prove robust is an open question.

Value

An object of `ctd-class`. The details of the contents depend on the source file. The `metadata` slot is particularly variable across data formats, because the meta-information provided in those formats varies widely.

Author(s)

Dan Kelley

References

Information about ice-tethered profile data is provided at http://www.whoi.edu/page.do?pid=23096, which also provides a link for downloading data. Note that the present version only handles data in profiler-mode, not fixed-depth mode.

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

---

read.ctd.odf *Read a CTD file in ODF format*

Description

Read a CTD file in ODF format

Usage

```
read.ctd.odf(file, columns = NULL, station = NULL, missingValue,
  monitor = FALSE, debug = getOption("oceDebug"), processingLog, ...)
```

Arguments

file	A connection or a character string giving the name of the file to load. For read.ctd.sbe() and read.ctd.woce(), this may be a wildcard (e.g. "*.cnv" or "*.csv") in which case the return value is a vector containing CTD objects created by reading the files from list.files with pattern set to the specified wildcard pattern.
columns	An optional list that can be used to convert unrecognized data names to resultant variable names. This is used only by read.ctd.sbe and read.ctd.odf; see "Examples".
station	Optional character string containing an identifying name or number for the station. This can be useful if the routine cannot determine the name automatically, or if another name is preferred.
missingValue	Optional missing-value flag; data matching this value will be set to NA upon reading. If this is provided, then it overrules any missing-value flag found in the data. For Seabird (.cnv) files, there is usually no need to set missingValue, because it can be inferred from the header (typically as -9.990e-29). Set missingValue=NULL

	to turn off missing-value detection, even in .cnv files that contain missing-value codes in their headers. If missingValue is not specified, then an attempt is made to infer such a value from the data, by testing whether salinity and/or temperature has a minimum that is under -8 in value; this should catch common values in files, without false positives. A warning will be issued in this case, and a note inserted in the processing log of the return value.
monitor	Boolean, set to TRUE to provide an indication of progress. This is useful if filename is a wildcard.
debug	An integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed.
processingLog	If provided, the action item to be stored in the log. This is typically only provided for internal calls; the default that it provides is better for normal calls by a user.
...	additional arguments, passed to called routines.

Details

read.ctd.odf reads files stored in Ocean Data Format, used in some Canadian hydrographic databases.

Value

An object of ctd-class. The details of the contents depend on the source file. The metadata slot is particularly variable across data formats, because the meta-information provided in those formats varies widely.

Author(s)

Dan Kelley

References

The ODF format, used by the Canadian Department of Fisheries and Oceans, is described to some extent in the documentation for read.odf. It is not clear that ODF format is handled correctly in read.ctd.odf, or the more general function read.odf, because the format varies between some sample files the author has encountered in his research.

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method,

oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Other things related to odf data: ODF2oce, ODFNames2oceNames, [[,odf-method, [[<-,odf-method, odf-class, plot,odf-method, read.odf, subset,odf-method, summary,odf-method

---

`read.ctd.sbe` *Read a Seabird CTD File*

Description

Read a Seabird CTD File

Usage

```
read.ctd.sbe(file, columns = NULL, station = NULL, missingValue,
  monitor = FALSE, debug = getOption("oceDebug"), processingLog, ...)
```

Arguments

`file`	A connection or a character string giving the name of the file to load. For `read.ctd.sbe()` and `read.ctd.woce()`, this may be a wildcard (e.g. `"*.cnv"` or `"*.csv"`) in which case the return value is a vector containing CTD objects created by reading the files from `list.files` with `pattern` set to the specified wildcard pattern.
`columns`	An optional `list` that can be used to convert unrecognized data names to resultant variable names. This is used only by `read.ctd.sbe` and `read.ctd.odf`; see "Examples".
`station`	Optional character string containing an identifying name or number for the station. This can be useful if the routine cannot determine the name automatically, or if another name is preferred.
`missingValue`	Optional missing-value flag; data matching this value will be set to `NA` upon reading. If this is provided, then it overrules any missing-value flag found in the data. For Seabird (`.cnv`) files, there is usually no need to set `missingValue`, because it can be inferred from the header (typically as -9.990e-29). Set `missingValue=NULL` to turn off missing-value detection, even in `.cnv` files that contain missing-value codes in their headers. If `missingValue` is not specified, then an attempt is made to infer such a value from the data, by testing whether salinity and/or temperature has a minimum that is under -8 in value; this should catch common values in files, without false positives. A warning will be issued in this case, and a note inserted in the processing log of the return value.
`monitor`	Boolean, set to `TRUE` to provide an indication of progress. This is useful if `filename` is a wildcard.
`debug`	An integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed.

processingLog	If provided, the action item to be stored in the log. This is typically only provided for internal calls; the default that it provides is better for normal calls by a user.
...	additional arguments, passed to called routines.

Details

This function reads files stored in Seabird `.cnv` format. Note that these files can contain multiple sensors for a given field. For example, the file might contain a column named `t090C` for one temperature sensor and `t190C` for a second. The first will be denoted `temperature` in the `data` slot of the return value, and the second will be denoted `temperature1`. This means that the first sensor will be used in any future processing that accesses `temperature`. This is for convenience of processing, and it does not pose a limitation, because the data from the second sensor are also available as e.g. `x[["temperature1"]]`, where `x` is the name of the returned value. For the details of the mapping from `.cnv` names to `ctd` names, see `cnvName2oceName`.

The original data names as stored in `file` are stored within the `metadata` slot as `dataNamesOriginal`, and are displayed with `summary` alongside the numerical summary. See the Appendix VI of [2] for the meanings of these names (in the "Short Name" column of the table spanning pages 161 through 172).

Value

An object of `ctd-class`. The details of the contents depend on the source file. The `metadata` slot is particularly variable across data formats, because the meta-information provided in those formats varies widely.

A note on scales

The user may encounter data files with a variety of scales for temperature and salinity. Oce keeps track of these scales in the units it sets up for the stored variables. For example, if `A` is a CTD object, then `A[["temperatureUnit"]]$scale` is a character string that will indicate the scale. Modern-day data will have `"ITS-90"` for that scale, and old data may have `"IPTS-68"`. The point of saving the scale in this way is so that the various formulas that deal with water properties can account for the scale, e.g. converting from numerical values saved on the `"IPTS-68"` scale to the newer scale, using `T90fromT68` before doing calculations that are expressed in terms of the `"ITS-90"` scale. This is taken care of by retrieving temperatures with the accessor function, e.g. writing `A[["temperature"]]` will either retrieve the stored values (if the scale is ITS-90) or converted values (if the scale is IPTS-68). Even though this procedure should work, users who really care about the details of their data are well-advised to do a couple of tests after examining the first data line of their data file in an editor.

Note that reading a file that contains IPTS-68 temperatures produces a warning.

Author(s)

Dan Kelley

References

1. The Sea-Bird SBE 19plus profiler is described at `http://www.seabird.com/products/spec_sheets/19plusdata.htm`. Some more information is given in the Sea-Bird data-processing manual `http://www.seabird.com/document/sbe-data-processing-manual`.
2. A SBE data processing manual is at `http://www.seabird.com/document/sbe-data-processing-manual`.

See Also

Other things related to ctd data: `[[,ctd-method`, `[[<-,ctd-method`, `as.ctd`, `cnvName2oceName`, `ctd-class`, `ctdDecimate`, `ctdFindProfiles`, `ctdRaw`, `ctdTrim`, `ctd`, `handleFlags,ctd-method`, `oceNames2whpNames`, `oceUnits2whpUnits`, `plot,ctd-method`, `plotProfile`, `plotScan`, `plotTS`, `read.ctd.itp`, `read.ctd.odf`, `read.ctd.woce.other`, `read.ctd.woce`, `read.ctd`, `subset,ctd-method`, `summary,ctd-method`, `woceNames2oceNames`, `woceUnit2oceUnit`, `write.ctd`

Examples

```
f <- system.file("extdata", "ctd.cnv", package="oce")
## Read the file in the normal way
d <- read.ctd(f)
## Read an imaginary file, in which salinity is named 'salt'
d <- read.ctd(f, columns=list(
  salinity=list(name="salt", unit=list(expression(), scale="PSS-78"))))
```

---

read.ctd.woce — *Read a WOCE-type CTD file with First Word "CTD"*

Description

Read a WOCE-type CTD file with First Word "CTD"

Usage

```
read.ctd.woce(file, columns = NULL, station = NULL, missingValue,
  monitor = FALSE, debug = getOption("oceDebug"), processingLog, ...)
```

Arguments

`file`	A connection or a character string giving the name of the file to load. For `read.ctd.sbe()` and `read.ctd.woce()`, this may be a wildcard (e.g. `"*.cnv"` or `"*.csv"`) in which case the return value is a vector containing CTD objects created by reading the files from `list.files` with `pattern` set to the specified wildcard pattern.
`columns`	An optional `list` that can be used to convert unrecognized data names to resultant variable names. This is used only by `read.ctd.sbe` and `read.ctd.odf`; see “Examples”.

station
: Optional character string containing an identifying name or number for the station. This can be useful if the routine cannot determine the name automatically, or if another name is preferred.

missingValue
: Optional missing-value flag; data matching this value will be set to `NA` upon reading. If this is provided, then it overrules any missing-value flag found in the data. For Seabird (`.cnv`) files, there is usually no need to set `missingValue`, because it can be inferred from the header (typically as -9.990e-29). Set `missingValue=NULL` to turn off missing-value detection, even in `.cnv` files that contain missing-value codes in their headers. If `missingValue` is not specified, then an attempt is made to infer such a value from the data, by testing whether salinity and/or temperature has a minimum that is under -8 in value; this should catch common values in files, without false positives. A warning will be issued in this case, and a note inserted in the processing log of the return value.

monitor
: Boolean, set to `TRUE` to provide an indication of progress. This is useful if `filename` is a wildcard.

debug
: An integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed.

processingLog
: If provided, the action item to be stored in the log. This is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

...
: additional arguments, passed to called routines.

Details

`read.ctd.woce()` reads files stored in the exchange format used by the World Ocean Circulation Experiment (WOCE), in which the first 4 characters are "`CTD,`". It also also in a rarer format with the first 3 characters are `CTD`" followed by a blank or the end of the line.

Value

An object of `ctd-class`. The details of the contents depend on the source file. The `metadata` slot is particularly variable across data formats, because the meta-information provided in those formats varies widely.

Author(s)

Dan Kelley

References

The WOCE-exchange format is described at `http://woce.nodc.noaa.gov/woce_v3/wocedata_1/whp/exchange/exchan` and a sample file is at `https://www.nodc.noaa.gov/woce/woce_v3/wocedata_1/whp/exchange/example_ct1.csv`

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

---

read.ctd.woce.other — *Read a WOCE-type CTD file with First Word "EXPOCODE"*

Description

Read a WOCE-type CTD file with First Word "EXPOCODE"

Usage

```
read.ctd.woce.other(file, columns = NULL, station = NULL, missingValue,
  monitor = FALSE, debug = getOption("oceDebug"), processingLog, ...)
```

Arguments

file	A connection or a character string giving the name of the file to load. For read.ctd.sbe() and read.ctd.woce(), this may be a wildcard (e.g. "*.cnv" or "*.csv") in which case the return value is a vector containing CTD objects created by reading the files from list.files with pattern set to the specified wildcard pattern.
columns	An optional list that can be used to convert unrecognized data names to resultant variable names. This is used only by read.ctd.sbe and read.ctd.odf; see "Examples".
station	Optional character string containing an identifying name or number for the station. This can be useful if the routine cannot determine the name automatically, or if another name is preferred.
missingValue	Optional missing-value flag; data matching this value will be set to NA upon reading. If this is provided, then it overrules any missing-value flag found in the data. For Seabird (.cnv) files, there is usually no need to set missingValue, because it can be inferred from the header (typically as -9.990e-29). Set missingValue=NULL to turn off missing-value detection, even in .cnv files that contain missing-value codes in their headers. If missingValue is not specified, then an attempt is made to infer such a value from the data, by testing whether salinity and/or temperature has a minimum that is under -8 in value; this should catch common values in files, without false positives. A warning will be issued in this case, and a note inserted in the processing log of the return value.
monitor	Boolean, set to TRUE to provide an indication of progress. This is useful if filename is a wildcard.

debug	An integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed.
processingLog	If provided, the action item to be stored in the log. This is typically only provided for internal calls; the default that it provides is better for normal calls by a user.
...	additional arguments, passed to called routines.

Details

read.ctd.woce.other() reads files stored in the exchange format used by the World Ocean Circulation Experiment (WOCE), in which the first word in the file is EXPOCODE.

Value

An object of ctd-class. The details of the contents depend on the source file. The metadata slot is particularly variable across data formats, because the meta-information provided in those formats varies widely.

Author(s)

Dan Kelley

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

---

read.echosounder *Read an Echosounder File*

Description

Reads a biosonics echosounder file. This function was written for and tested with single-beam, dual-beam, and split-beam Biosonics files of type V3, and may not work properly with other file formats.

Usage

```
read.echosounder(file, channel = 1, soundSpeed, tz = getOption("oceTz"),
  debug = getOption("oceDebug"), processingLog)
```

Arguments

`file`	a connection or a character string giving the name of the file to load.
`channel`	sequence number of channel to extract, for multi-channel files.
`soundSpeed`	sound speed, in m/s. If not provided, this is calculated using `swSoundSpeed(35, 15, 30, eos="unesco")` (In theory, it could be calculated using the temperature and salinity that are stored in the data file, but these will just be nominal values, anyway.
`tz`	character string indicating time zone to be assumed in the data.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`processingLog`	if provided, the action item to be stored in the log, typically only provided for internal calls.

Value

An object of `class` `"echosounder"` with standard slots `metadata`, `data` and `processingLog` that are described in the documentation for the object `echosounder-class`.

Bugs

Only the amplitude information (in counts) is determined. A future version of this function may provide conversion to dB, etc. The handling of dual-beam and split-beam files is limited. In the dual-beam cse, only the wide beam signal is processed (I think ... it could be the narrow beam, actually, given the confusing endian tricks being played). In the split-beam case, only amplitude is read, with the x-axis and y-axis angle data being ignored.

Author(s)

Dan Kelley, with help from Clark Richards

References

Various echosounder instruments provided by BioSonics are described at the company website, `http://www.biosonicsinc.com/`. The document listed as [1] below was provided to the author of this function in November 2011, which suggests that the data format was not changed since July 2010.

[1] Biosonics, 2010. DT4 Data File Format Specification. BioSonics Advanced Digital Hydroacoustics. July, 2010. SOFTWARE AND ENGINEERING LIBRARY REPORT BS&E-2004-07-0009-2.0.

See Also

The documentation for `echosounder-class` explains the structure of `ctd` objects, and also outlines the other functions dealing with them.

Other things related to `echosounder` data: `[[,echosounder-method`, `[[<-,echosounder-method`, `as.echosounder`, `echosounder-class`, `echosounder`, `findBottom`, `plot,echosounder-method`, `subset,echosounder-method`, `summary,echosounder-method` read.g1sst *Read a G1SST file*

Description

Read a G1SST file in the netcdf format provided by the ERDAPP server [1].

Usage

```
read.g1sst(filename)
```

Arguments

filename	name of a netcdf file containing G1SST data.

Details

As noted in the documentation for g1sst-class, one must be aware of the incorporation of model simulations in the g1sst product. For example, the code presented below might lead one to believe that the mapped field represents observations, whereas in fact it can be verified by consulting [2] (clicking and unclicking the radio button to show just the data) that the field mostly derives from simulation.

Value

An object of g1sst-class.

Author(s)

Dan Kelley

References

1. ERDDAP Portal http://coastwatch.pfeg.noaa.gov/erddap/ 2. JPO OurOcean Portal http://ourocean.jpl.nasa. (link worked in 2016 but was seen to fail 2017 Feb 2).

See Also

Other things related to satellite data: g1sst-class, plot,satellite-method, satellite-class, summary,satellite-method

Examples

```
## Not run:
# Construct query, making it easier to understand and modify.
day <- "2016-01-02"
lon0 <- -66.5
lon1 <- -64.0
lat0 <- 44
```

```
lat1 <- 46
source <- paste("http://coastwatch.pfeg.noaa.gov/erddap/griddap/",
                "jplG1SST.nc?",
                "SST%5B(", day, "T12:00:00Z)",
                "%5D%5B(", lat0, "):(", lat1, ")",
                "%5D%5B(", lon0, "):(", lon1, ")",
                "%5D", sep="")
if (!length(list.files(pattern="^a.nc$")))
    download.file(source, "a.nc")
d <- read.g1sst("a.nc")
plot(d, "SST", col=oceColorsJet)
data(coastlineWorldFine, package="ocedata")
lines(coastlineWorldFine[['longitude']],coastlineWorldFine[['latitude']])

## End(Not run)
```

---

read.gps *Read a GPS File*

Description

Reads GPX format files by simply finding all longitudes and latitudes; in other words, information on separate tracks, or waypoints, etc., is lost.

Usage

```
read.gps(file, type = NULL, debug = getOption("oceDebug"), processingLog)
```

Arguments

`file`	name of file containing gps data.
`type`	type of file, which will be inferred from examination of the data if not supplied. In the present version, the only choice for `type` is `"gpx"`.
`debug`	set to TRUE to print information about the header, etc.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Value

An object of `class` `"gps"`.

Author(s)

Dan Kelley

See Also

Other things related to gps data: `[[,gps-method`, `[[<-,gps-method`, `as.gps`, `gps-class`, `plot,gps-method`, `summary,gps-method` read.index *Read a NOAA ocean index file*

Description

Read an ocean index file, in the format used by NOAA.

Usage

```
read.index(file, format, missingValue, tz = getOption("oceTz"),
  debug = getOption("oceDebug"))
```

Arguments

`file`	a connection or a character string giving the name of the file to load. May be a URL.
`format`	optional character string indicating the format type. If not supplied, a guess will be made, based on examination of the first few lines of the file. If supplied, the possibilities are `"noaa"` and `"ucar"`. See "Details".
`missingValue`	If supplied, this is a numerical value that indicates invalid data. In some datasets, this is -99.9, but other values may be used. If `missingValue` is not supplied, any data that have value equal to or less than -99 are considered invalid. Set `missingValue` to `TRUE` to turn of the processing of missing values.
`tz`	character string indicating time zone to be assumed in the data.
`debug`	a flag that turns on debugging, ignored in the present version of the function.

Details

Reads a text-format index file, in either of two formats. If `format` is missing, then the first line of the file is examined. If that line contains 2 (whitespace-separated) tokens, then `"noaa"` format is assumed. If it contains 13 tokens, then `"ucar"` format is assumed. Otherwise, an error is reported.

In the `"noaa"` format, the two tokens on the first line are taken to be the start year and the end year, respectively. The second line must contain a single token, the missing value. All further lines must contain 12 tokens, for the values in January, February, etc.

In the `"ucar"` format, all data lines must contain the 13 tokens, the first of which is the year, with the following 12 being the values for January, February, etc.

Value

A data frame containing `t`, a POSIX time, and `index`, the numerical index. The times are set to the 15th day of each month, which is a guess that may need to be changed if so indicated by documentation (yet to be located).

Author(s)

Dan Kelley

References

See https://www.esrl.noaa.gov/psd/data/climateindices/list/ for a list of indices.

Examples

```
## Not run:
library(oce)
par(mfrow=c(2, 1))
# 1. AO, Arctic oscillation
ao <- read.index("https://www.esrl.noaa.gov/psd/data/correlation/ao.data")
aorecent <- subset(ao, t > as.POSIXct("2000-01-01"))
oce.plot.ts(aorecent$t, aorecent$index)
# 2. SOI, probably more up-to-date then data(soi, package="ocedata")
soi <- read.index("https://www.cgd.ucar.edu/cas/catalog/climind/SOI.signal.ascii")
soirecent <- subset(soi, t > as.POSIXct("2000-01-01"))
oce.plot.ts(soirecent$t, soirecent$index)

## End(Not run)
```

`read.landsat` *Read a landsat File Directory*

Description

Read a landsat data file, producing an object of `landsat-class`. The actual reading is done with readTIFF in the **tiff** package, so that package must be installed for `read.landsat` to work.

Usage

```
read.landsat(file, band = "all", emissivity = 0.984, decimate,
  debug = getOption("oceDebug"))
```

Arguments

`file`	A connection or a character string giving the name of the file to load. This is a directory name containing the data.
`band`	The bands to be read, by default all of the bands. Use 'band=NULL' to skip the reading of bands, instead reading only the image metadata, which is often enough to check if the image is of interest in a given study. See 'Details' for the names of the bands, some of which are pseudo-bands, computed from the actual data.
`emissivity`	Value of the emissivity of the surface, stored as `emissivity` in the `metadata` slot of the resultant object. This is used in the calculation of surface temperature, as explained in the discussion of accessor functions for `landsat-class`. The default value is from Konda et al. (1994). These authors suggest an uncertainty of 0.04, but a wider range of values can be found in the literature. The value of `metadata$emissivity` is easy to alter, either as a single value or as a matrix, yielding flexibility of calculation.

decimate	optional positive integer indicating the degree to which the data should be subsampled after reading and before storage. Setting this to 10 can speed up reading by a factor of 3 or more, but higher values have diminishing effect. In exploratory work, it is useful to set decimate=10, to plot the image to determine a subregion of interest, and then to use landsatTrim to trim the image.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

Landsat data are provided in directories that contain TIFF files and header information, and read.landsat relies on a strict convention for the names of the files in those directories. Those file names were found by inspection of some data, on the assumption that similar patterns will hold for other datasets for any given satellite. This is a brittle approach and it should be born in mind if read.landsat fails for a given dataset.

For Landsat 8, there are 11 bands, with names "aerosol" (band 1), "blue" (band 2), "green" (band 3), "red" (band 4), "nir" (band 5), "swir1" (band 6), "swir2" (band 7), "panchromatic" (band 8), "cirrus" (band 9), "tirs1" (band 10), and "tirs2" (band 11). In addition to the above, setting band="terralook" may be used as an abbreviation for band=c("red", "green", "nir").

For Landsat 7, there 8 bands, with names "blue" (band 1), "green" (band 2), "red" (band 3), "nir" (band 4), "swir1" (band 5), "tir1" (band 6A), "tir2" (band 6B), "swir2" (band 7) and "panchromatic" (band 8).

For Landsat 4 and 5, the bands similar to Landsat 7 but without "panchromatic" (band 8).

Value

An object of landsat-class, with the conventional Oce slots metadata, data and processingLog. The metadata is mainly intended for use by Oce functions, but for generality it also contains an entry named header that represents the full image header in a list (with names made lower-case). The data slot holds matrices of the data in the requested bands, and users may add extra matrices if desired, e.g. to store calculated quantities.

Storage requirements

Landsat data files (directories, really) are large, accounting for approximately 1 gigabyte each. The storage of the Oce object is similar (see landsat-class). In R, many operations involving copying data, so that dealing with full-scale landsat images can overwhelm computers with storage under 8GB. For this reason, it is typical to read just the bands that are of interest. It is also helpful to use landsatTrim to trim the data to a geographical range, or to use decimate to get a coarse view of the domain, especially early in an analysis.

Author(s)

Dan Kelley

References

1. Konda, M. Imasato N., Nishi, K., and T. Toda, 1994. Measurement of the Sea Surface Emissivity. *Journal of Oceanography*, 50, 17:30. Available at `http://www.terrapub.co.jp/journals/JO/pdf/5001/50010017.pdf` as of February 2015.

See Also

`landsat-class` for more information on `landsat` objects, especially band information. Use `landsatTrim` to trim Landsat objects geographically and `landsatAdd` to add new "bands." The accessor operator (`[[`) is used to access band information, full or decimated, and to access certain derived quantities. A sample dataset named `landsat` is provided by the **oce** package.

Other things related to `landsat` data: `[[,landsat-method`, `landsat-class`, `landsatAdd`, `landsatTrim`, `landsat`, `plot,landsat-method`, `summary,landsat-method`

`read.lisst` *Read a LISST File*

Description

Read a LISST data file, producing a `lisst` object, i.e. one inheriting from `lisst-class`. The file should contain 42 columns, with no header. If there are fewer than 42 columns, an error results. If there are more, only the first 42 are used. Note that `read.oce` can recognize LISST files by their having a name ending in `".asc"` and by having 42 elements on the first line. Even so, it is preferred to use the present function, because it gives the opportunity to specify the year and timezone, so that times can be calculated properly.

Usage

```
read.lisst(file, year = 0, tz = "UTC", longitude = NA, latitude = NA)
```

Arguments

`file`	a connection or a character string giving the name of the file to load.
`year`	year in which the measurement of the series was made.
`tz`	time zone.
`longitude`	longitude of observation (stored in metadata)
`latitude`	latitude of observation (stored in metadata)

Value

An object of `lisst-class`.

Author(s)

Dan Kelley

See Also

Other things related to lisst data: [[,lisst-method, [[<-,lisst-method, as.lisst, lisst-class, plot,lisst-method, summary,lisst-method

---

read.lobo *Read a LOBO File*

Description

Read a data file created by a LOBO instrument.

Usage

```
read.lobo(file, cols = 7, processingLog)
```

Arguments

file	a connection or a character string giving the name of the file to load.
cols	number of columns in dataset.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Details

This version of read.lobo is really quite crude, having been developed mainly for a "predict the Spring bloom" contest at Dalhousie University. In particular, the function assumes that the data columns are exactly as specified in the Examples section; if you reorder the columns or add new ones, this function is unlikely to work correctly. Furthermore, it should be noted that the file format was inferred simply by downloading files; the supplier makes no claims that the format will be fixed in time. It is also worth noting that there is no read.oce equivalent to read.lobo, because the file format has no recognizable header.

Value

An object of lobo-class.

Author(s)

Dan Kelley

Examples

```
## Not run:
library(oce)
uri <- paste("http://lobo.satlantic.com/cgi-bin/nph-data.cgi?",
  "min_date=20070220&max_date=20070305",
  "&x=date&",
  "y=current_across1,current_along1,nitrate,fluorescence,salinity,temperature&",
  "data_format=text",sep="")
lobo <- read.lobo(uri)

## End(Not run)
```

`read.met` *Read a met File*

Description

Reads a comma-separated value file in the format used by the Environment Canada [1]. The agency does not publish a format for these files, so this function was based on a study of a few sample files, and it may fail for other files, if Environment Canada changes the format.

Usage

```
read.met(file, type = NULL, skip, tz = getOption("oceTz"),
  debug = getOption("oceDebug"))
```

Arguments

`file`	a connection or a character string giving the name of the file to load.
`type`	if `NULL`, then the first line is studied, in order to determine the file type. If `type="msc"`, then a file as formatted by Environment Canada is assumed.
`skip`	optional number of lines of header that occur before the actual data. If this is not supplied, `read.met` scans the file until it finds a line starting with `"Date/Time"`, and considers all lines above that to be header.
`tz`	timezone assumed for time data
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Value

An object of `class` `"met"`, of which the `data` slot contains vectors `time`, `temperature`, `pressure`, `u`, and `v`. The velocity components have units m/s and are the components of the vector of wind direction. In other words, the oceanographic convention on velocity is employed, not the meteorological one; the weather forecaster's "North wind" has positive `v` and zero `u`. In addition to these things, `data` also contains `wind` (in km/h), taken straight from the data file.

Note

There seem to be several similar formats in use, so this function may not work in all cases.

Author(s)

Dan Kelley

References

1. Environment Canada website for Historical Climate Data http://climate.weather.gc.ca/index_e.html

See Also

Other things related to met data: [[,met-method, [[<-,met-method, as.met, download.met, met-class, met, plot,met-method, subset,met-method, summary,met-method

Examples

```
## Not run:
library(oce)
# Recreate data(met) and plot u(t) and v(t)
metFile <- download.met(id=6358, year=2003, month=9, destdir=".")
met <- read.met(metFile)
met <- oceSetData(met, "time", met[["time"]]+4*3600,
                  note="add 4h to local time to get UTC time")
plot(met, which=3:4)

## End(Not run)
```

---

read.netcdf *Read a NetCDF File*

Description

Read a NetCDF File

Usage

```
read.netcdf(file, ...)
```

Arguments

file	the name of a file
...	unused

Details

Read a netcdf file, trying to interpret its contents sensibly.

It is important to note that this is a preliminary version of this function, and much about it may change without notice. Indeed, it may be removed entirely.

Below are some features that may be changed.

1. The names of data items are not changed from those in the netcdf file on the assumption that this will offer the least surprise to the user.

2. An attempt is made to find some common metadata from global attributes in the netcdf file. These attributes include `Longitude`, `Latitude`, `Ship` and `Cruise`. Before they are stored in the metadata, they are converted to lower case, since that is the oce convention.

Value

An object of `oce-class`.

---

read.oce — *Read an Oceanographic Data File*

Description

Read an oceanographic data file, auto-discovering the file type from the first line of the file. This function tries to infer the file type from the first line, using `oceMagic`. If it can be discovered, then an instrument-specific file reading function is called, with the `file` and with any additional arguments being supplied.

Usage

```
read.oce(file, ...)
```

Arguments

`file`	a connection or a character string giving the name of the file to load.
`...`	arguments to be handed to whichever instrument-specific reading function is selected, based on the header.

Value

An object of `oce-class` that is specialized to the data type, e.g. `ctd-class`, if the data file contains `ctd` data.

Author(s)

Dan Kelley

See Also

The file type is determined by `oceMagic`. If the file type can be determined, then one of the following is called: `read.ctd`, `read.coastline` `read.lobo`, `read.rsk`, `read.sealevel`, etc.

Examples

```
library(oce)
x <- read.oce(system.file("extdata", "ctd.cnv", package="oce"))
plot(x) # summary with TS and profiles
plotTS(x) # just the TS
```

`read.odf` *Read an ODF file*

Description

ODF (Ocean Data Format) is a format developed at the Bedford Institute of Oceanography and also used at other Canadian Department of Fisheries and Oceans (DFO) facilities. It can hold various types of time-series data, which includes a variety of instrument types. Thus, `read.odf` is used by `read.ctd.odf` for CTD data, etc. As of mid-2015, `read.odf` is still in development, with features being added as a project with DFO makes available more files.

Usage

```
read.odf(file, columns = NULL, debug = getOption("oceDebug"))
```

Arguments

`file`	the file containing the data.
`columns`	An optional `list` that can be used to convert unrecognized data names to resultant variable names. For example, `columns=list(salinity=list(name="salt", unit=list(unit=ex` states that a short-name of `"salt"` represents salinity, and that the unit is as indicated. This is passed to `cnvName2oceName` or `ODFNames2oceNames`, as appropriate, and takes precedence over the lookup table in that function.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

Note that some elements of the metadata are particular to ODF objects, e.g. depthMin, depthMax and sounding, which are inferred from ODF items named MIN_DEPTH, MAX_DEPTH and SOUNDING, respectively. In addition, the more common metadata item waterDepth, which is used in ctd objects to refer to the total water depth, is set to sounding if that is finite, or to maxDepth otherwise.

The function ODFNames2oceNames is used to translate data names from the ODF file to standard oce names, and handles conversion for a few non-standard units. The documentation of ODFNames2oceNames should be consulted for more details.

Value

An object of class oce. It is up to a calling function to determine what to do with this object.

Caution

ODF files do not store information on the temperature or salinity scale, and read.odf assumes them to be ITS-90 and PSS-78, respectively. These scales will not be correct for old data files. Note that the temperature scale can be converted from old scales using T90fromT68 and T90fromT48, although the change will be in a fraction of a millidegree, which probably exceeds reasonable confidence in old data.

References

[1] Anthony W. Isenor and David Kellow, 2011. ODF Format Specification Version 2.0. (This is a .doc file downloaded from a now-forgotten URL by Dan Kelley, in June 2011.)

[2] The St Lawrence Global Observatory website has information on ODF format at https://slgo.ca/app-sgdo/en/docs_reference/format_odf.html

See Also

ODF2oce will be an alternative to this, once (or perhaps if) a ODF package is released by the Canadian Department of Fisheries and Oceans.

Other things related to odf data: ODF2oce, ODFNames2oceNames, [[,odf-method, [[<-,odf-method, odf-class, plot,odf-method, read.ctd.odf, subset,odf-method, summary,odf-method

Examples

```
library(oce)
# Read a CTD cast made on the Scotian Shelf. Note that the file's metadata
# states that conductivity is in S/m, but it is really conductivity ratio,
# so we must alter the unit before converting to a CTD object. Note that
# read.odf() on this data file produces a warning suggesting that the user
# repair the unit, using the method outlined here.
odf <- read.odf(system.file("extdata", "CTD_BCD2014666_008_1_DN.ODF", package="oce"))
odf[["conductivityUnit"]] <- list(unit=expression(), scale="")
#
# Figure 1. make a CTD, and plot (with span to show NS)
plot(as.ctd(odf), span=500)
# Figure 2. highlight bad data on TS diagram
```

```
plotTS(odf, type='o') # use a line to show loops
bad <- odf[["QCFlag"]]!=0
points(odf[['salinity']][bad],odf[['temperature']][bad],col='red',pch=20)
```

---

read.rsk *Read a Rsk file*

Description

Read an RBR rsk or txt file, e.g. as produced by an RBR logger, producing an object of class `rsk`.

Usage

```
read.rsk(file, from = 1, to, by = 1, type, tz = getOption("oceTz", default
  = "UTC"), patm = FALSE, processingLog, debug = getOption("oceDebug"))
```

Arguments

`file`	a connection or a character string giving the name of the file to load. Note that `file` must be a character string, because connections are not used in that case, which is instead handled with database calls.
`from`	indication of the first datum to read. This can a positive integer to indicate sequence number, the POSIX time of the first datum, or a character string that can be converted to a POSIX time. (For POSIX times, be careful about the `tz` argument.)
`to`	an indication of the last datum to be read, in the same format as `from`. If `to` is missing, data will be read to the end of the file.
`by`	an indication of the stride length to use while walking through the file. If this is an integer, then `by-1` samples are skipped between each pair of samples that is read. If this is a string representing a time interval, in colon-separated format (HH:MM:SS or MM:SS), then this interval is divided by the sampling interval, to get the stride length.
`type`	optional file type, presently can be `rsk` or `txt` (for a text export of an RBR rsk or hex file). If this argument is not provided, an attempt will be made to infer the type from the file name and contents.
`tz`	time zone. The value `oceTz` is set at package setup.
`patm`	controls the handling of atmospheric pressure, an important issue for RBR instruments that record absolute pressure; see "Details".
`processingLog`	if provided, the action item to be stored in the log. This is typically only provided for internal calls; the default that it provides is better for normal calls by a user.
`debug`	a flag that can be set to `TRUE` to turn on debugging.

Details

This can read files produced by several RBR instruments. At the moment, five styles are understood: (1) text file produced as an export of an RBR `hex` or `rsk` file; (2) text file with columns for temperature and pressure (with sampling times indicated in the header); (3) text file with four columns, in which the date the time of day are given in the first two columns, followed by the temperature, and pressure; (4) text file with five columns, in which depth in the water column is given after the pressure; (5) an SQLite-based database format. The first four options are provided mainly for historical reasons, since RBR instruments at the date of writing commonly use the SQLite format, though the first option is common for all instruments that produce a `hex` file that can be read using Ruskin.

Options 2-4 are mostly obsolete, and will be removed from future versions.

A note on conductivity. RBR devices record conductivity in mS/cm, and it is this value that is stored in the object returned by `read.rsk`. This can be converted to conductivity ratio (which is what many other instruments report) by dividing by 42.914 (see Culkin and Smith, 1980) which will be necessary in any seawater-related function that takes conductivity ratio as an argument (see "Examples").

A note on pressure. RBR devices tend to record absolute pressure (i.e. sea pressure plus atmospheric pressure), unlike most oceanographic instruments that record sea pressure (or an estimate thereof). The handling of pressure is controlled with the `patm` argument, for which there are three possibilities. (1) If `patm` is `FALSE` (the default), then pressure read from the data file is stored in the `data` slot of return value, and the `metadata` item `pressureType` is set to the string `"absolute"`. (2) If `patm` is `TRUE`, then an estimate of atmospheric pressure is subtracted from the raw data. For data files in the SQLite format (i.e. `*.rsk` files), this estimate will be the value read from the file, or the "standard atmosphere" value 10.1325 dbar, if the file lacks this information. (3) If `patm` is a numerical value (or list of values, one for each sampling time), then `patm` is subtracted from the raw data. In cases 2 and 3, an additional column named `pressureOriginal` is added to the `data` slot to store the value contained in the data file, and `pressureType` is set to a string starting with `"sea"`. See `as.ctd` for details of how this setup facilitates the conversion of `rsk-class` objects to `ctd-class` objects.

Value

An object of `rsk-class`.

Author(s)

Dan Kelley and Clark Richards

References

Culkin, F., and Norman D. Smith, 1980. Determination of the concentration of potassium chloride solution having the same electrical conductivity, at 15 C and infinite frequency, as standard seawater of salinity 35.0000 ppt (Chlorinity 19.37394 ppt). *IEEE Journal of Oceanic Engineering*, **5**, pp 22-23.

See Also

The documentation for `rsk-class` explains the structure of `rsk` objects, and also outlines other functions dealing with them. Since RBR has a wide variety of instruments, `rsk` datasets can be quite general, and it is common to coerce rsk objects to other forms for specialized work, e.g. as.ctd can be used to create CTD object, so that the generic plot obeys the CTD format.

Other things related to rsk data: [[,rsk-method, [[<-,rsk-method, as.rsk, plot,rsk-method, rsk-class, rskPatm, rskToc, rsk, subset,rsk-method, summary,rsk-method

---

read.sealevel *Read a Sealevel File*

Description

Read a data file holding sea level data. BUG: the time vector assumes GMT, regardless of the GMT.offset value.

Usage

```
read.sealevel(file, tz = getOption("oceTz"), processingLog,
  debug = getOption("oceDebug"))
```

Arguments

file	a connection or a character string giving the name of the file to load. See Details for the types of files that are recognized.
tz	time zone. The default value, oceTz, is set to UTC at setup. (If a time zone is present in the file header, this will supercede the value given here.)
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

This function starts by scanning the first line of the file, from which it determines whether the file is in one of two known formats: type 1, the format used at the Hawaii archive centre, and type 2, the comma-separated-value format used by the Marine Environmental Data Service. (The file type is inferred by checking for the existence of the string Station_Name on the first line of the file, indicating type 2.) If the file is in neither of these formats, the user might wish to scan it directly, and then to use as.sealevel to create a sealevel object.

Value

An object of sealevel-class.

Author(s)

Dan Kelley

References

The Hawaii archive site at `http://ilikai.soest.hawaii.edu/uhslc/datai.html` provides a graphical interface for downloading sealevel data in Type 1, with format as described at `http://ilikai.soest.hawaii.edu/` (this link worked for years but failed at least temporarily on December 4, 2016). The MEDS repository (`http://www.isdm-gdsi.gc.ca/isdm-gdsi/index-eng.html`) provides Type 2 data.

See Also

Other things related to sealevel data: `[[,sealevel-method`, `[[<-,sealevel-method`, `as.sealevel`, `plot,sealevel-method`, `sealevel-class`, `sealevelTuktoyaktuk`, `sealevel`, `subset,sealevel-method`, `summary,sealevel-method`

Examples

```
## Not run:
library(oce)
# this yields the sealevel dataset
sl <- read.oce("h275a96.dat")
summary(sl)
plot(sl)
m <- tidem(sl)
plot(m)

## End(Not run)
```

`read.section` *Read a Section File*

Description

Read a file that contains a series of `ctd` profiles that make up an oceanographic section. Only *exchange BOT* comma-separated value format is permitted at this time, but other formats may be added later. It should also be noted that the parsing scheme was developed after inspection of the A03 data set (see Examples). This may cause problems if the format is not universal. For example, the header must name the salinity column `"CTDSAL"`; if not, salinity values will not be read from the file.

Usage

```
read.section(file, directory, sectionId = "", flags, ship = "",
  scientist = "", institute = "", missingValue = -999,
  debug = getOption("oceDebug"), processingLog)
```

Arguments

`file`	A file containing a set of CTD observations. At present, only the *exchange BOT* format is accepted (see Details).
`directory`	A character string indicating the name of a directory that contains a set of CTD files that hold individual stations in the section.
`sectionId`	Optional string indicating the name for the section. If not provided, the section ID is determined by examination of the file header.
`flags`	Ignored, and deprecated (will be disallowed in a future version).
`ship`	Name of the ship carrying out the sampling.
`scientist`	Name of chief scientist aboard ship.
`institute`	Name of chief scientist's institute.
`missingValue`	Numerical value used to indicate missing data.
`debug`	Logical. If `TRUE`, print some information that might be helpful during debugging.
`processingLog`	If provided, the action item to be stored in the log. This is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

Value

An object of class `section-class`.

Disambiguating salinity

WOCE datasets commonly have a column named `CTDSAL` for salinity inferred from a CTD and `SALNTY` (not a typo) for salinity derived from bottle data. If only one of these is present in the data file, the data will be called `salinity` in the `data` slot of the return value. However, if both are present, then `CTDSAL` is stored as `salinity` and `SALNTY` is stored as `salinityBottle`.

Author(s)

Dan Kelley

References

Several repository sites provide section data. An example that is perhaps likely to exist for years is `https://cchdo.ucsd.edu`, but a search on `"WOCE bottle data"` should turn up other sites, if this one ceases to exist. Only the so-called *exchange BOT* data format can be processed by read.section() at this time. Data names are inferred from column headings using `woceNames2oceNames`.

See Also

Other things related to `section` data: `[[,section-method`, `[[<-,section-method`, `as.section`, `handleFlags,section-method`, `plot,section-method`, `section-class`, `sectionAddStation`, `sectionGrid`, `sectionSmooth`, `sectionSort`, `section`, `subset,section-method`, `summary,section-method` read.topo *Read a Topo File*

Description

Read a file that contains topographic data in the ETOPO dataset, as provided by the NOAA website (see `download.topo` for a good server for such files.

Usage

```
read.topo(file, debug = getOption("oceDebug"))
```

Arguments

`file`	Name of a file containing an ETOPO-format dataset. Three types are permitted; see “Details”.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

The three permitted file types are as follows: (1) an ascii type described by NOAA as "?"; (2) a NetCDF format described by NOAA as "GMT NetCDF" (recognized by the presence of a variable named), and (3) another NetCDF format described by NOAA as "NetCDF" (recognized by the presence of a variable called). Files in each of these formats can be downloaded with `download.topo`.

Value

An object of type `topo-class` that which has the following slots.

`data`	: a data frame containing `lat`, `lon`, and `z`
`metadata`	: a list containing the source filename
`processingLog`	: a log, in the standard `oce` format.

Author(s)

Dan Kelley

See Also

Other things related to `topo` data: `[[,topo-method`, `[[<-,topo-method`, `as.topo`, `download.topo`, `plot,topo-method`, `subset,topo-method`, `summary,topo-method`, `topo-class`, `topoInterpolate`, `topoWorld`

Examples

```
## Not run:
library(oce)
topoMaritimes <- read.topo("topoMaritimes.asc")
plot(topographyMaritimes)

## End(Not run)
```

---

renameData — *Rename items in the data slot of an oce object*

Description

This function may be used to rename elements within the `data` slot of `oce` objects. It also updates the processing log of the returned object, indicating the changes.

Usage

```
renameData(x, old = NULL, new = NULL)
```

Arguments

`x`	An oce object, i.e. one inheriting from `oce-class`.
`old`	Vector of strings, containing old names.
`new`	Vector of strings, containing old names.

Examples

```
data(ctd)
new <- renameData(ctd, "temperature", "temperature68")
new <- oceSetData(new, name="temperature",
                  value=T90fromT68(new[["temperature68"]]),
                  unit=list(unit=expression(degree*C),scale="ITS=90"))
```

---

rescale — *Rescale values to lie in a given range*

Description

This is helpful in e.g. developing a colour scale for an image plot. It is not necessary that `rlow` be less than `rhigh`, and in fact reversing them is a good way to get a reversed colour scale for a plot.

Usage

```
rescale(x, xlow, xhigh, rlow = 0, rhigh = 1, clip = TRUE)
```

Arguments

x	a numeric vector.
xlow	x value to correspond to rlow. If not given, it will be calculated as the minimum value of x
xhigh	x value to correspond to rhigh. If not given, it will be calculated as the maximum value of x
rlow	value of the result corresponding to x equal to xlow.
rhigh	value of the result corresponding to x equal to xhigh.
clip	logical, set to TRUE to clip the result to the range spanned by rlow and rhigh.

Value

A new vector, which has minimum lim[1] and maximum lim[2].

Author(s)

Dan Kelley

Examples

```
library(oce)
# Fake tow-yow data
t <- seq(0, 600, 5)
x <- 0.5 * t
z <- 50 * (-1 + sin(2 * pi * t / 360))
T <- 5 + 10 * exp(z / 100)
palette <- oce.colorsJet(100)
zlim <- range(T)
drawPalette(zlim=zlim, col=palette)
plot(x, z, type='p', pch=20, cex=3,
     col=palette[rescale(T, xlow=zlim[1], xhigh=zlim[2], rlow=1, rhigh=100)])
```

---

resizableLabel *Provide axis names in adjustable sizes*

Description

Provide axis names in adjustable sizes, e.g. using T instead of Temperature, and including units as appropriate. Used by e.g. plot,ctd-method.

Usage

```
resizableLabel(item, axis, sep, unit = NULL)
```

Arguments

item	code for the label. This must be an element from the following list, or an abbreviation that uniquely identifies an element through its first letters: "S", "C", "conductivity mS/cm", "conductivity S/m", "T", "theta", "sigmaTheta", "conservative temperature", "absolute salinity", "nitrate", "nitrite", "oxygen", "oxygen saturation", "oxygen mL/L", "oxygen umol/L", "oxygen umol/kg", "phosphate", "silicate", "tritium", "spice", "fluorescence", "p", "z", "distance", "distance km", "along-track distance km", "heading", "pitch", "roll", "u", "v", "w", "speed", "direction", "eastward", "northward", "depth", "elevation", "latitude", "longitude", "frequency cph", or "spectral density m2/cph".
axis	a string indicating which axis to use; must be x or y.
sep	optional character string inserted between the unit and the parentheses or brackets that enclose it. If not provided, then getOption("oceUnitSep") is checked. If that exists, then it is used as the separator; if not, no separator is used.
unit	optional unit to use, if the default is not satisfactory. This might be the case if for example temperature was not measured in Celcius.

Value

A character string or expression, in either a long or a shorter format, for use in the indicated axis at the present plot size. Whether the unit is enclosed in parentheses or square brackets is determined by the value of getOption("oceUnitBracket"), which may be "[" or "(". Whether spaces are used between the unit and these deliminators is set by psep or getOption("oceUnitSep").

Author(s)

Dan Kelley

---

retime *Adjust the time within Oce object*

Description

This function compensates for drifting instrument clocks, according to $t' = t + a + b(t - t0)$, where t' is the true time and t is the time stored in the object. A single check on time mismatch can be described by a simple time offset, with a non-zero value of a, a zero value of b, and an arbitrary value of t0. Checking the mismatch before and after an experiment yields sufficient information to specify a linear drift, via a, b, and t0. Note that t0 is just a convenience parameter, which avoids the user having to know the "zero time" used in R and clarifies the values of the other two parameters. It makes sense for t0 to have the same timezone as the time within x.

Usage

```
retime(x, a, b, t0, debug = getOption("oceDebug"))
```

Arguments

x	an oce object (presently, this must be of class adv)
a	intercept [in seconds] in linear model of time drift (see “Details”).
b	slope [unitless] in linear model of time drift [unitless] (see “Details”).
t0	reference time [in POSIXct format] used in linear model of time drift (see “Details”).
debug	a flag that, if nonzero, turns on debugging.

Details

The returned object is computed by linear interpolation, using approx with rule=2, to avoid NA values at the start or end. The new time will be as given by the formula above. Note that if the drift is large enough, the sampling rate will be changed. It is a good idea to start with an object that has an extended time range, so that, after this is called, subset can be used to trim to a desired time range.

Value

A new object, with time and other data adjusted.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(adv)
adv2 <- retime(adv,0,1e-4,as.POSIXct("2008-07-01 00:00:00",tz="UTC"))
plot(adv[["time"]], adv2[["time"]]-adv[["time"]], type='l')
```

---

rsk *Sample Rsk Dataset*

Description

A sample rsk object derived from a Concerto CTD manufactured by RBR Ltd.

Details

The data were obtained September 2015, off the west coast of Greenland, by Matt Rutherford and Nicole Trenholm of the Ocean Research Project, in collaboration with RBR and with the NASA Oceans Melting Greenland project.

References

https://rbr-global.com/

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to `rsk` data: `[[,rsk-method`, `[[<-,rsk-method`, `as.rsk`, `plot,rsk-method`, `read.rsk`, `rsk-class`, `rskPatm`, `rskToc`, `subset,rsk-method`, `summary,rsk-method`

Examples

```
library(oce)
data(rsk)
## The object doesn't "know" it is CTD until told so
plot(rsk)
plot(as.ctd(rsk))
```

`rsk-class` *Class to Store Rsk Data*

Description

Class for data stored in the "Ruskin" format used by RBR [1], including both `rsk` SQLite files and the ASCII `txt` exported files.

A `rsk` object may be read with `read.rsk` or created with `as.rsk`. Plots can be made with `plot,rsk-method`, while `summary,rsk-method` produces statistical summaries and `show` produces overviews. If atmospheric pressure has not been removed from the data, the functions `rskPatm` may provide guidance as to its value; however, this last function is no equal to decent record-keeping at sea. Data may be retrieved with `[[,rsk-method` or replaced with `[[<-,rsk-method`.

Author(s)

Dan Kelley and Clark Richards

References

1. RBR website: www.rbr-global.com/products

See Also

Other classes provided by oce: `adp-class`, `adv-class`, `argo-class`, `bremen-class`, `cm-class`, `coastline-class`, `ctd-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `oce-class`, `odf-class`, `sealevel-class`, `section-class`, `topo-class`, `windrose-class`

Other things related to `rsk` data: `[[,rsk-method`, `[[<-,rsk-method`, `as.rsk`, `plot,rsk-method`, `read.rsk`, `rskPatm`, `rskToc`, `rsk`, `subset,rsk-method`, `summary,rsk-method` rsk2ctd *Create a ctd Object from an rsk Object*

Description

A new ctd object is assembled from the contents of the rsk object. The data and metadata are mostly unchanged, with an important exception: the pressure item in the data slot may altered, because rsk instruments measure total pressure, not sea pressure; see "Details".

Usage

```
rsk2ctd(x, pressureAtmospheric = 0, longitude = NULL, latitude = NULL,
  ship = NULL, cruise = NULL, station = NULL, deploymentType = NULL,
  debug = getOption("oceDebug"))
```

Arguments

x	An rsk object, i.e. one inheriting from rsk-class.
pressureAtmospheric	A numerical value (a constant or a vector), that is subtracted from the pressure in object before storing it in the return value.
longitude	numerical value of longitude, in degrees East.
latitude	numerical value of latitude, in degrees North.
ship	optional string containing the ship from which the observations were made.
cruise	optional string containing a cruise identifier.
station	optional string containing a station identifier.
deploymentType	character string indicating the type of deployment (see as.ctd).
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

The pressureType element of the metadata of rsk objects defines the pressure type, and this controls how pressure is set up in the returned object. If object@metadata$pressureType is "absolute" (or NULL) then the resultant pressure will be adjusted to make it into "sea" pressure. To do this, the value of object@metadata$pressureAtmospheric is inspected. If this is present, then it is subtracted from pressure. If this is missing, then standard pressure (10.1325 dbar) will be subtracted. At this stage, the pressure should be near zero at the ocean surface, but some additional adjustment might be necessary, and this may be indicated by setting the argument pressureAtmospheric to a non-zero value to be subtracted from pressure.

rskPatm *Estimate Atmospheric Pressure in Rsk Object*

Description

Estimate atmospheric pressure in rsk record.

Usage

```
rskPatm(x, dp = 0.5)
```

Arguments

x	A rsk object, or a list of pressures (in decibars).
dp	Half-width of pressure window to be examined (in decibars).

Details

Pressures must be in decibars for this to work. First, a subset of pressures is created, in which the range is sap-dp to sap+dp. Here, sap=10.1325 dbar is standard sealevel atmospheric pressure. Within this window, three measures of central tendency are calculated: the median, the mean, and a weighted mean that has weight given by $exp(-2 * ((p - sap)/dp)^2)$.

Value

A list of four estimates: sap, the median, the mean, and the weighted mean.

Author(s)

Dan Kelley

See Also

The documentation for rsk-class explains the structure of rsk objects, and also outlines the other functions dealing with them.

Other things related to rsk data: [[,rsk-method, [[<-,rsk-method, as.rsk, plot,rsk-method, read.rsk, rsk-class, rskToc, rsk, subset,rsk-method, summary,rsk-method

Examples

```
library(oce)
data(rsk)
print(rskPatm(rsk))
```

rskToc *Decode table-of-contents File from a Rsk File*

Description

Decode table-of-contents file from a rsk file, of the sort used by some researchers at Dalhousie University.

Usage

```
rskToc(dir, from, to, debug = getOption("oceDebug"))
```

Arguments

`dir`	name of a directory containing a single table-of-contents file, with `.TBL` at the end of its file name.
`from`	optional `POSIXct` time, indicating the beginning of a data interval of interest. This must have timezone `"UTC"`.
`to`	optional `POSIXct` time, indicating the end of a data interval of interest. This must have timezone `"UTC"`.
`debug`	optional integer to control debugging, with positive values indicating to print information about the processing.

Details

It is assumed that the `.TBL` file contains lines of the form `"File  \day179\SL08A179.023 started at Fri Jun 27 22:00:` The first step is to parse these lines to get day and hour information, i.e. 179 and 023 in the line above. Then, recognizing that it is common to change the names of such files, the rest of the filename information in the line is ignored, and instead a new file name is constructed based on the data files that are found in the directory. (In other words, the `"\day179\SL08A"` portion of the line is replaced.) Once the file list is complete, with all times put into R format, then (optionally) the list is trimmed to the time interval indicated by `from` and `to`. It is important that `from` and `to` be in the `UTC` time zone, because that time zone is used in decoding the lines in the `.TBL` file.

Value

A list with two elements: `filename`, a vector of file names, and `startTime`, a vector of `POSIXct` times indicating the (real) times of the first datum in the corresponding files.

Author(s)

Dan Kelley

See Also

Other things related to `rsk` data: `[[,rsk-method`, `[[<-,rsk-method`, `as.rsk`, `plot,rsk-method`, `read.rsk`, `rsk-class`, `rskPatm`, `rsk`, `subset,rsk-method`, `summary,rsk-method`

Examples

```
## Not run:
table <- rskToc("/data/archive/sleiwex/2008/moorings/m05/adv/sontek_202h/raw",
from=as.POSIXct("2008-07-01 00:00:00", tz="UTC"),
    to=as.POSIXct("2008-07-01 12:00:00", tz="UTC"))
print(table)

## End(Not run)
```

---

runlm *Calculate running linear models*

Description

The linear model is calculated from the slope of a localized least-squares regression model y=y(x). The localization is defined by the x difference from the point in question, with data at distance exceeding L/2 being ignored. With a `boxcar` window, all data within the local domain are treated equally, while with a `hanning` window, a raised-cosine weighting function is used; the latter produces smoother derivatives, which can be useful for noisy data. The function is based on internal calculation, not on `lm`.

Usage

```
runlm(x, y, xout, window = c("hanning", "boxcar"), L, deriv)
```

Arguments

`x`	a vector holding x values.
`y`	a vector holding y values.
`xout`	optional vector of x values at which the derivative is to be found. If not provided, `x` is used.
`window`	type of weighting function used to weight data within the window; see 'Details'.
`L`	width of running window, in x units. If not provided, a reasonable default will be used.
`deriv`	an optional indicator of the desired return value; see 'Examples'.

Value

If `deriv` is not specified, a list containing vectors of output values `y` and `y`, derivative (`dydx`), along with the scalar length scale `L`. If `deriv=0`, a vector of values is returned, and if `deriv=1`, a vector of derivatives is returned.

Author(s)

Dan Kelley

Examples

```
library(oce)

# Case 1: smooth a noisy signal
x <- 1:100
y <- 1 + x/100 + sin(x/5)
yn <- y + rnorm(100, sd=0.1)
L <- 4
calc <- runlm(x, y, L=L)
plot(x, y, type='l', lwd=7, col='gray')
points(x, yn, pch=20, col='blue')
lines(x, calc$y, lwd=2, col='red')

# Case 2: square of buoyancy frequency
data(ctd)
par(mfrow=c(1,1))
plot(ctd, which="N2")
rho <- swRho(ctd)
z <- swZ(ctd)
zz <- seq(min(z), max(z), 0.1)
N2 <- -9.8 / mean(rho) * runlm(z, rho, zz, deriv=1)
lines(N2, -zz, col='red')
legend("bottomright", lwd=2, bg="white",
       col=c("black", "red"),
       legend=c("swN2()", "using runlm()"))
```

---

satellite-class   *Class to hold satellite data*

Description

This class holds satellite data of various types, including amsr-class and g1sst-class.

Author(s)

Dan Kelley and Chantelle Layton

See Also

Other things related to satellite data: g1sst-class, plot,satellite-method, read.g1sst, summary,satellite-method `sealevel` *Sealevel data for Halifax Harbour*

Description

This sample sea-level dataset is the 2003 record from Halifax Harbour in Nova Scotia, Canada. For reasons that are not mentioned on the data archive website, the record ends on the 8th of October.

Author(s)

Dan Kelley

Source

The data were created as

```
 sealevel <-
read.oce("490-01-JAN-2003_slev.csv") sealevel <- oce.edit(sealevel,
"longitude", -sealevel[["longitude"]], reason="Fix longitude hemisphere")
```

where the csv file was downloaded from [1]. Note the correction of longitude sign, which is required because the data file has no indication that this is the western hemisphere.

References

1. Fisheries and Oceans Canada http://www.meds-sdmm.dfo-mpo.gc.ca/isdm-gdsi/index-eng.html

See Also

Other datasets provided with oce: adp, adv, argo, cm, coastlineWorld, colors, ctdRaw, ctd, echosounder, landsat, lisst, lobo, met, rsk, sealevelTuktoyaktuk, section, topoWorld, wind

Other things related to sealevel data: [[,sealevel-method, [[<-,sealevel-method, as.sealevel, plot,sealevel-method, read.sealevel, sealevel-class, sealevelTuktoyaktuk, subset,sealevel-method, summary,sealevel-method

sealevel-class — *Class to Store Sealevel Data*

Description

Class to store sealevel data, e.g. from a tide gauge, with standard slots `metadata`, `data` and `processingLog`.

Methods

Data may be accessed as e.g. `sealevel[["time"]]`, where the string could also be e.g. `"elevation"` for the corresponding sea-level elevation, or e.g. `"longitude"` or `"latitude"` for scalars. Items in `metadata` must be specifield by full name, but those in `data` may be abbreviated, so long as the abbreviation is unique.

Everything that may be accessed may also be assigned, e.g. `sealevel[["elevation"]] <- value`.

The `show` method displays information about the object, while `summary,sealevel-method` provides a statistical summary.

Author(s)

Dan Kelley

See Also

Other classes provided by `oce`: `adp-class`, `adv-class`, `argo-class`, `bremen-class`, `cm-class`, `coastline-class`, `ctd-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `oce-class`, `odf-class`, `rsk-class`, `section-class`, `topo-class`, `windrose-class`

Other things related to `sealevel` data: `[[,sealevel-method`, `[[<-,sealevel-method`, `as.sealevel`, `plot,sealevel-method`, `read.sealevel`, `sealevelTuktoyaktuk`, `sealevel`, `subset,sealevel-method`, `summary,sealevel-method`

sealevelTuktoyaktuk — *Sea-level data set acquired in 1975 at Tuktoyaktuk*

Description

This sea-level dataset is provided with in Appendix 7.2 of Foreman (1977) and also with the `T_TIDE` package (Pawlowicz et al., 2002). It results from measurements made in 1975 at Tuktoyaktuk, Northwest Territories, Canada.

Details

The data set contains 1584 points, some of which have NA for sea-level height.

Although Foreman's Appendix 7.2 states that times are in Mountain standard time, the timezone is set to `UTC` in the present case, so that the results will be similar to those he provides in his Appendix 7.3.

Historical note

Until Jan 6, 2018, the time in this dataset had been increased by 7 hours. However, this alteration was removed on this date, to make for simpler comparison of amplitude and phase output with the results obtained by Foreman (1977) and Pawlowicz et al. (2002).

Source

The data were based on the `T_TIDE` dataset, which in turn seems to be based on Appendix 7.2 of Foreman (1977). Minor editing was on file format, and then the `sealevelTuktoyaktuk` object was created using `as.sealevel`.

References

Foreman, M. G. G., 1977. Manual for tidal heights analysis and prediction. Pacific Marine Science Report 77-10, Institute of Ocean Sciences, Patricia Bay, Sidney, BC, 58pp.

Pawlowicz, Rich, Bob Beardsley, and Steve Lentz, 2002. Classical tidal harmonic analysis including error estimates in MATLAB using `T_TIDE`. Computers and Geosciences, 28, 929-937.

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevel`, `section`, `topoWorld`, `wind`

Other things related to sealevel data: `[[,sealevel-method`, `[[<-,sealevel-method`, `as.sealevel`, `plot,sealevel-method`, `read.sealevel`, `sealevel-class`, `sealevel`, `subset,sealevel-method`, `summary,sealevel-method`

Examples

```
## Not run:
library(oce)
data(sealevelTuktoyaktuk)
time <- sealevelTuktoyaktuk[["time"]]
elevation <- sealevelTuktoyaktuk[["elevation"]]
oce.plot.ts(time, elevation, type='l', ylab="Height [m]", ylim=c(-2, 6))
legend("topleft", legend=c("Tuktoyaktuk (1975)","Detided"),
       col=c("black","red"),lwd=1)
tide <- tidem(sealevelTuktoyaktuk)
detided <- elevation - predict(tide)
lines(time, detided, col="red")

## End(Not run)
```

`secondsToCtime` *Time interval as colon-separated string*

Description

Convert a time interval to a colon-separated string

Usage

```
secondsToCtime(sec)
```

Arguments

`sec`	length of time interval in seconds.

Value

A string with a colon-separated time interval.

Author(s)

Dan Kelley

See Also

See `ctimeToSeconds`, the inverse of this.

Other things related to time: `ctimeToSeconds`, `julianCenturyAnomaly`, `julianDay`, `numberAsHMS`, `numberAsPOSIXct`, `unabbreviateYear`

Examples

```
library(oce)
cat("   10 s = ", secondsToCtime(10), "\n", sep="")
cat("   61 s = ", secondsToCtime(61), "\n", sep="")
cat("86400 s = ", secondsToCtime(86400), "\n", sep="")
```

section *Hydrographic section*

Description

This is line A03 (ExpoCode 90CT40_1, with nominal sampling date 1993-09-11). The chief scientist was Tereschenkov of SOI, working aboard the Russian ship Multanovsky, undertaking a westward transect from the Mediterranean outflow region across to North America, with a change of heading in the last few dozen stations to run across the nominal Gulf Stream axis. The data flags follow the WHP CTD convention, i.e. 1 for uncalibrated, 2 for an acceptable measurement, 3 for a questionable measurement, 4 for a bad measurement, etc; see `https://www.nodc.noaa.gov/woce/woce_v3/wocedata_1/whp/exchange/exchange_format_desc.htm` for further details.

Usage

```
data(section)
```

Source

This is based on the WOCE file named `a03_hy1.csv`, downloaded from `https://cchdo.ucsd.edu/cruise/90CT40_1`, 13 April 2015.

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `topoWorld`, `wind`

Other things related to section data: `[[,section-method`, `[[<-,section-method`, `as.section`, `handleFlags,section-method`, `plot,section-method`, `read.section`, `section-class`, `sectionAddStation`, `sectionGrid`, `sectionSmooth`, `sectionSort`, `subset,section-method`, `summary,section-method`

Examples

```
## Not run:
library(oce)
# Gulf Stream
data(section)
GS <- subset(section, 109<=stationId&stationId<=129)
GSg <- sectionGrid(GS, p=seq(0, 5000, 100))
plot(GSg, map.xlim=c(-80,-60))

## End(Not run)
```

section-class *Class to Store Hydrographic Section Data*

Description

Class to store hydrographic section data, with standard slots `metadata`, `data` and `processingLog`.

A `list` of stations is retrieved by `s[["station"]]`. Individual stations are retrieved by providing a station number as a second argument in the index, e.g. the first station is `s[["station", 1]]` (which is a `ctd-class` object).

Aggregated values of the quantities measured at each level of the CTD profiles contained within the section may be accessed as e.g. `section[["salinity"]]`. This works for any quantity whose name is present in the constituent profiles.

Since it is often useful to pair such quantities with locations, `section[["longitude"]]` and `section[["latitude"]]` return vectors with values repeated for each level in each CTD (see the `pairs()` call in the example section). If just one latitude or longitude is desired per station, e.g. `section[["latitude", "byStation"]]` may be used. Station-by-station values of dynamic height are provided by e.g. `section[["dynamic height"]]`.

The depths of all data are obtained from e.g. `section[["depth"]]`, and the distances along the transect, measured from the first station, are obtained from e.g. `section[["distance"]]`.

Author(s)

Dan Kelley

See Also

Sections can be read with `read.section` or created with `read.section` or created from CTD objects by using `as.section` or by adding a ctd station to an existing section with `sectionAddStation`.

Sections may be sorted with `sectionSort`, subsetted with `subset,section-method`, smoothed with `sectionSmooth`, and gridded with `sectionGrid`. Gridded sections may be plotted with `plot,section-method`.

Statistical summaries are provided by `summary,section-method`, while overviews are provided by `show`.

The sample dataset `section` contains data along WOCE line A03.

Other classes provided by oce: `adp-class`, `adv-class`, `argo-class`, `bremen-class`, `cm-class`, `coastline-class`, `ctd-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `oce-class`, `odf-class`, `rsk-class`, `sealevel-class`, `topo-class`, `windrose-class`

Other things related to section data: `[[,section-method`, `[[<-,section-method`, `as.section`, `handleFlags,section-method`, `plot,section-method`, `read.section`, `sectionAddStation`, `sectionGrid`, `sectionSmooth`, `sectionSort`, `section`, `subset,section-method`, `summary,section-method`

Examples

```
library(oce)
data(section)
plot(section[['station', 1]])
pairs(cbind(z=-section[["pressure"]],T=section[["temperature"]],S=section[["salinity"]]))
## T profiles for first few stations in section, at common scale
par(mfrow=c(3,3))
Tlim <- range(section[["temperature"]])
ylim <- rev(range(section[["pressure"]]))
for (stn in section[["station", 1:9]])
    plotProfile(stn, xtype='potential temperature', ylim=ylim, Tlim=Tlim)
```

`sectionAddStation` *Add a CTD Station to a Section*

Description

Add a CTD profile to an existing section.

Usage

```
sectionAddStation(section, station)
```

Arguments

`section`	A section to which a station is to be added.
`station`	A ctd object holding data for the station to be added.

Value

An object of `class` `section`.

Historical note

Until March 2015, this operation was carried out with the + operator, but at that time, the syntax was flagged by the development version of R, so it was changed to the present form.

Author(s)

Dan Kelley

See Also

Other things related to section data: `[[,section-method`, `[[<-,section-method`, `as.section`, `handleFlags,section-method`, `plot,section-method`, `read.section`, `section-class`, `sectionGrid`, `sectionSmooth`, `sectionSort`, `section`, `subset,section-method`, `summary,section-method`

Examples

```
library(oce)
data(ctd)
ctd2 <- ctd
ctd2[["temperature"]] <- ctd2[["temperature"]] + 0.5
ctd2[["latitude"]] <- ctd2[["latitude"]] + 0.1
section <- as.section(c("ctd", "ctd2"))
ctd3 <- ctd
ctd3[["temperature"]] <- ctd[["temperature"]] + 1
ctd3[["latitude"]] <- ctd[["latitude"]] + 0.1
ctd3[["station"]] <- "Stn 3"
sectionAddStation(section, ctd3)
```

---

sectionGrid *Grid a Section*

Description

Grid a section, by interpolating to fixed pressure levels. The "approx", "boxcar" and "lm" methods are described in the documentation for ctdDecimate, which is used to do this processing. The default "approx" method is best for bottle data, the "boxcar" is best for ctd data, and the "lm" method is probably too slow to recommend for exploratory work, in which it is common to do trials with a variety of "p" values.

Usage

```
sectionGrid(section, p, method = "approx", debug = getOption("oceDebug"),
  ...)
```

Arguments

section	A section object containing the section to be gridded.
p	Optional indication of the pressure levels to which interpolation should be done. If this is not supplied, the pressure levels will be calculated based on the typical spacing in the ctd profiles stored within section. If p="levitus", then pressures will be set to be those of the Levitus atlas, given by standardDepths, trimmed to the maximum pressure in section. If p is a single numerical value, it is taken as the number of subdivisions to use in a call to seq that has range from 0 to the maximum pressure in section. Finally, if a vector numerical values is provided, then it is used as is.
method	The method to use to decimate data within the stations; see ctdDecimate, which is used for the decimation.
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces

the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

... Optional arguments to be supplied to ctdDecimate, e.g. rule controls extrapolation beyond the observed pressure range, in the case where method equals "approx".

Value

An object of section-class that contains stations whose pressure values match identically.

A note about flags

Data-quality flags contained within the original object are ignored by this function, and the returned value contains no such flags. This is because such flags represent an assessment of the original data, not of quantities derived from those data. This function produces a warning to this effect. The recommended practice is to use handleFlags or some other means to deal with flags before calling the present function.

Author(s)

Dan Kelley

See Also

Other things related to section data: [[,section-method, [[<-,section-method, as.section, handleFlags,section-method, plot,section-method, read.section, section-class, sectionAddStation, sectionSmooth, sectionSort, section, subset,section-method, summary,section-method

Examples

```
# Gulf Stream
library(oce)
data(section)
GS <- subset(section, 109<=stationId&stationId<=129)
GSg <- sectionGrid(GS, p=seq(0, 5000, 100))
plot(GSg, map.xlim=c(-80,-60))
```

---

sectionSmooth *Smooth a Section*

Description

Smooth a section in the lateral (alpha version that may change).

Usage

```
sectionSmooth(section, method = c("spline", "barnes"), xg, yg, xgl, ygl, xr,
  yr, gamma = 0.5, iterations = 2, trim = 0, pregrid = FALSE,
  debug = getOption("oceDebug"), ...)
```

Arguments

`section`	A `section` object containing the section to be smoothed. For `method="spline"`, the pressure levels must match for each station in the section.
`method`	Specifies the method to use; see 'Details'.
`xg, xgl`	passed to `interpBarnes`, if `method="barnes"`; ignored otherwise. If `xg` is supplied, it defines the x component of the grid, i.e. the resultant "stations". Alternatively, if `xgl` is supplied, the x grid is established using `seq`, to span the data with `xgl` elements. If neither of these is supplied, the output x grid will equal the input x grid.
`yg, ygl`	similar to `xg` and `xgl`.
`xr, yr`	influence ranges in x and y, passed to `interpBarnes` if `method="barnes"`; ignored otherwise.
`gamma`	scale-reduction parameter, passed to `interpBarnes`, if `method="barnes"`; ignored otherwise.
`iterations`	number of iterations of Barnes algorithm, passed to `interpBarnes`, if `method="barnes"`; ignored otherwise.
`trim`	passed to `interpBarnes`, if `method="barnes"`; ignored otherwise
`pregrid`	passed to `interpBarnes`, if `method="barnes"`; ignored otherwise
`debug`	A flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	Optional extra arguments, passed to either `smooth.spline` or `interpBarnes`.

Details

This function should be used with caution, as should any operation that changes data. Although smoothing may be desirable to produce aesthetically-pleasing plots, it can also introduce artifacts that can lead to erroneous conclusions. The prudent analyst starts by comparing plots of the raw data with plots of the smoothed data.

For `method="spline"`, the section is smoothed using `smooth.spline` on individual pressure levels, with any parameters listed in `parameters` being passed to that function. If `df` is not present in `parameters`, then this function sets it to the number of stations divided by 5. Smoothing is done separately for temperature, salinity, and sigma-theta.

For the (much slower) `method="barnes"` method, smoothing is done across both horizontal and vertical coordinates, using `interpBarnes`. The stations are changed to lie on the grid supplied defined `xg` and `yg`, or by `xgl` and `ygl` (see those arguments)

Value

An object of `section-class` that ordered in some way.

Author(s)

Dan Kelley

See Also

Other things related to section data: [[,section-method, [[<-,section-method, as.section, handleFlags,section-method, plot,section-method, read.section, section-class, sectionAddStation, sectionGrid, sectionSort, section, subset,section-method, summary,section-method

Examples

```
library(oce)
data(section)
gs <- subset(section, 109<=stationId&stationId<=129)
gsg <- sectionGrid(gs, p=seq(0, 5000, 150))
gss1 <- sectionSmooth(gsg, "spline", df=16)
plot(gss1)
## Not run:
gss2 <- sectionSmooth(gsg, "barnes", xr=24, yr=100)
plot(gss2)

## End(Not run)
```

---

sectionSort *Sort a Section*

Description

Sections created with as.section have "stations" that are in the order of the CTD objects (or filenames for such objects) provided. Sometimes, this is not the desired order, e.g. if file names discovered with dir are in an order that makes no sense. (For example, a practioner might name stations "stn1", "stn2", etc., not realizing that this will yield an unhelpful ordering, by file name, if there are more than 9 stations.) The purpose of sectionSort is to permit reordering the constituent stations in sensible ways.

Usage

```
sectionSort(section, by)
```

Arguments

section	A section object containing the section whose stations are to be sorted.
by	An optional string indicating how to reorder. If not provided, "stationID" will be assumed. Other choices are "distance", for distance from the first station, "longitude", for longitude, "latitude" for latitude, and "time", for time.

Value

An object of section-class that has less lateral variation than the input section.

Author(s)

Dan Kelley

See Also

Other things related to section data: [[,section-method, [[<-,section-method, as.section, handleFlags,section-method, plot,section-method, read.section, section-class, sectionAddStation, sectionGrid, sectionSmooth, section, subset,section-method, summary,section-method

Examples

```
## Not run:
# Eastern North Atlantic, showing Mediterranean water;
# sorting by longitude makes it easier to compare
# the map and the section.
library(oce)
data(section)
s <- sectionGrid(subset(section, -30 <= longitude))
ss <- sectionSort(ss, by="longitude")
plot(ss)

## End(Not run)
```

---

shiftLongitude *Shift Longitude to Range -180 to 180*

Description

This is a utility function used by mapGrid. It works simply by subtracting 180 from each longitude, if any longitude in the vector exceeds 180.

Usage

```
shiftLongitude(longitudes)
```

Arguments

longitudes	a numerical vector of longitudes

Value

vector of longitudes, shifted to the desired range.

See Also

matrixShiftLongitude and standardizeLongitude.

Other functions related to maps: lonlat2map, lonlat2utm, map2lonlat, mapArrows, mapAxis, mapContour, mapDirectionField, mapGrid, mapImage, mapLines, mapLocator, mapLongitudeLatitudeXY, mapPlot, mapPoints, mapPolygon, mapScalebar, mapText, mapTissot, oceCRS, usrLonLat, utm2lonlat

---

showMetadataItem *Show metadata item*

Description

This is a helper function for various summary functions.

Usage

```
showMetadataItem(object, name, label = "", postlabel = "", isdate = FALSE,
  quote = FALSE)
```

Arguments

object	an object inheriting from the base oce class.
name	name of item
label	label to print before item
postlabel	label to print after item
isdate	boolean indicating whether the item is a time
quote	boolean indicating whether to enclose the item in quotes

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
showMetadataItem(ctd, "ship", "ship")
```

siderealTime *Convert a POSIXt time to a sidereal time*

Description

Convert a POSIXt time to a sidereal time, using the method in Chapter 7 of Meeus (1982). The small correction that he discusses after his equation 7.1 is not applied here.

Usage

```
siderealTime(t)
```

Arguments

t	a time, in POSIXt format, e.g. as created by as.POSIXct, as.POSIXlt, or numberAsPOSIXct. If this is provided, the other arguments are ignored.

Value

A sidereal time, in hours in the range from 0 to 24.

Author(s)

Dan Kelley

References

Meeus, Jean, 1982. Astronomical formuae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages

See Also

Other things related to astronomy: eclipticalToEquatorial, equatorialToLocalHorizontal, julianCenturyAnomaly, julianDay, moonAngle, sunAngle

Examples

```
t <- ISOdatetime(1978, 11, 13, 0, 0, 0, tz="UTC")
print(siderealTime(t))
```

standardDepths *Standard Oceanographic Depths*

Description

This returns so-called standard depths 0m, 10m, etc. below the sea surface.

Usage

```
standardDepths()
```

Value

A vector of depths, c(0, 10, ...).

Author(s)

Dan Kelley

standardizeLongitude *Put longitude in the range from -180 to 180*

Description

Put longitude in the range from -180 to 180

Usage

```
standardizeLongitude(longitude)
```

Arguments

`longitude`	in degrees East, possibly exceeding 180

Value

longitude in signed degrees East

See Also

`matrixShiftLongitude` and `shiftLongitude` are more powerful relatives to `standardizeLongitude`.

subset,adp-method *Subset an ADP Object*

Description

Subset an adp (acoustic Doppler profile) object, in a manner that is function is somewhat analogous to subset.data.frame. Subsetting can be by time or distance, but these may not be combined; use a sequence of calls to subset by both.

Usage

```
## S4 method for signature 'adp'
subset(x, subset, ...)
```

Arguments

x	An adp object, i.e. one inheriting from adp-class.
subset	A condition to be applied to the data portion of x. See 'Details'.
...	Ignored.

Value

A new adp-class object.

Author(s)

Dan Kelley

See Also

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

Other functions that subset oce objects: subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-method, subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(adp)
# First part of time series
plot(subset(adp, time < mean(range(adp[['time']]))))
```

subset,adv-method *Subset an ADV Object*

Description

Subset an adv (acoustic Doppler profile) object. This function is somewhat analogous to subset.data.frame, except that subsets can only be specified in terms of time.

Usage

```
## S4 method for signature 'adv'
subset(x, subset, ...)
```

Arguments

x	An adv object, i.e. one inheriting from adv-class.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Value

A new adv-class object.

Author(s)

Dan Kelley

See Also

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv-class, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

Other functions that subset oce objects: subset,adp-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-method, subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(adv)
plot(adv)
plot(subset(adv, time < mean(range(adv[['time']]))))
```

subset,amsr-method *Subset an amsr Object*

Description

This function is somewhat analogous to subset.data.frame, but only one independent variable may be used in subset in any call to the function, which means that repeated calls will be necessary to subset based on more than one independent variable (e.g. latitude and longitude).

Usage

```
## S4 method for signature 'amsr'
subset(x, subset, ...)
```

Arguments

x	A amsr object, i.e. one inheriting from amsr-class.
subset	An expression indicating how to subset x.
...	Ignored.

Value

An amsr object.

Author(s)

Dan Kelley

See Also

Other things related to amsr data: [[<-,amsr-method, amsr-class, composite,amsr-method, download.amsr, plot,amsr-method, read.amsr, summary,amsr-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-method, subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
## Not run:
library(oce)
earth <- read.amsr("f34_20160102v7.2.gz") # not provided with oce
fclat <- subset(earth , 45<=latitude & latitude <= 49)
fc <- subset(fclat , longitude <= -47 & longitude <= -43)
plot(fc)

## End(Not run)
```

subset,argo-method *Subset an Argo Object*

Description

Subset an argo object, either by selecting just the "adjusted" data or by subsetting by pressure or other variables.

Usage

```
## S4 method for signature 'argo'
subset(x, subset, ...)
```

Arguments

x	An object inheriting from argo-class.
subset	An expression indicating how to subset x.
...	Ignored.

Details

If subset is the string "adjusted", then subset replaces the station variables with their adjusted counterparts. In the argo notation, e.g. PSAL is replaced with PSAL_ADJUSTED; in the present notation, this means that salinity in the data slot is replaced with salinityAdjusted, and the latter is deleted. Similar replacements are also done with the flags stored in the metadata slot.

If subset is an expression, then the action is somewhat similar to other subset functions, but with the restriction that only one independent variable may be used in in any call to the function, so that repeated calls will be necessary to subset based on more than one independent variable. Subsetting may be done by anything stored in the data, e.g. time, latitude, longitude, profile, dataMode, or pressure or by profile (a made-up variable) or id (from the metadata slot).

Value

An argo object.

Author(s)

Dan Kelley

See Also

Other things related to argo data: [[,argo-method, [[<-,argo-method, argo-class, argoGrid, argoNames2oceNames, argo, as.argo, handleFlags,argo-method, plot,argo-method, read.argo, summary,argo-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-method, subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(argo)

# Example 1: buset by time, longitude, and pressure
par(mfrow=c(2,2))
plot(argo)
plot(subset(argo, time > mean(time)))
plot(subset(argo, longitude > mean(longitude)))
plot(subset(argoGrid(argo), pressure > 500 & pressure < 1000), which=5)

# Example 2: restrict attention to delayed-mode profiles.
par(mfrow=c(1, 1))
plot(subset(argo, dataMode == "D"))

# Example 3: contrast corrected and uncorrected data
par(mfrow=c(1,2))
plotTS(argo)
plotTS(subset(argo, "adjusted"))
```

`subset,cm-method` *Subset a CM Object*

Description

This function is somewhat analogous to `subset.data.frame`.

Usage

```
## S4 method for signature 'cm'
subset(x, subset, ...)
```

Arguments

`x`	a `cm` object, i.e. inheriting from `cm-class`.
`subset`	a condition to be applied to the `data` portion of `x`. See 'Details'.
`...`	ignored.

Value

A new `cm` object.

Author(s)

Dan Kelley

See Also

Other things related to cm data: `[[,cm-method`, `[[<-,cm-method`, `as.cm`, `cm-class`, `cm`, `plot,cm-method`, `read.cm`, `summary,cm-method`

Other functions that subset oce objects: `subset,adp-method`, `subset,adv-method`, `subset,amsr-method`, `subset,argo-method`, `subset,coastline-method`, `subset,ctd-method`, `subset,echosounder-method`, `subset,lobo-method`, `subset,met-method`, `subset,oce-method`, `subset,odf-method`, `subset,rsk-method`, `subset,sealevel-method`, `subset,section-method`, `subset,topo-method`

Examples

```
library(oce)
data(cm)
plot(cm)
plot(subset(cm, time < mean(range(cm[['time']]))))
```

---

subset,coastline-method *Subset a Coastline Object*

Description

Summarizes coastline length, bounding box, etc.

Usage

```
## S4 method for signature 'coastline'
subset(x, subset, ...)
```

Arguments

`x`	A coastline object, i.e. one inheriting from `coastline-class`.
`subset`	An expression indicating how to subset `x`.
`...`	Ignored.

Value

A `coastline` object.

Author(s)

Dan Kelley

See Also

Other things related to coastline data: [[,coastline-method, [[<-,coastline-method, as.coastline, coastline-class, coastlineBest, coastlineCut, coastlineWorld, download.coastline, plot,coastline-method, read.coastline.openstreetmap, read.coastline.shapefile, summary,coastline-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,ctd-method, subset,echosounder-method, subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

---

subset,ctd-method *Subset a CTD Object*

Description

Return a subset of a section object.

Usage

```
## S4 method for signature 'ctd'
subset(x, subset, ...)
```

Arguments

x	A ctd object, i.e. one inheriting from ctd-class.
subset	An expression indicating how to subset x.
...	optional arguments, of which only the first is examined. The only possibility is that this argument be named indices. See “Details”.

Details

This function is used to subset data within the levels of a ctd object. There are two ways of working. If subset is supplied, then it is a logical expression that is evaluated within the environment of the data slot of the object (see Example 1). Alternatively, if the ... list contains an expression defining indices, then that expression is used to subset each item within the data slot (see Example 2).

Value

A ctd-class object.

Author(s)

Dan Kelley

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,echosounder-method, subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(ctd)
plot(ctd)
## Example 1
plot(subset(ctd, pressure<10))
## Example 2
plot(subset(ctd, indices=1:10))
```

---

subset,echosounder-method

Subset an Echosounder Object

Description

This function is somewhat analogous to subset.data.frame. Subsetting can be by time or depth, but these may not be combined; use a sequence of calls to subset by both.

Usage

```
## S4 method for signature 'echosounder'
subset(x, subset, ...)
```

Arguments

x	a echosounder object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Value

A new echosounder object.

Author(s)

Dan Kelley

See Also

Other things related to echosounder data: [[,echosounder-method, [[<-,echosounder-method, as.echosounder, echosounder-class, echosounder, findBottom, plot,echosounder-method, read.echosounder, summary,echosounder-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(echosounder)
plot(echosounder)
plot(subset(echosounder, depth < 10))
plot(subset(echosounder, time < mean(range(echosounder[['time']]))))
```

---

subset,lobo-method *Subset a LOBO Object*

Description

Subset an lobo object, in a way that is somewhat analogous to subset.data.frame.

Usage

```
## S4 method for signature 'lobo'
subset(x, subset, ...)
```

Arguments

x	a lobo object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Value

A new lobo object.

Author(s)

Dan Kelley

See Also

Other things related to lobo data: [[,lobo-method, [[<-,lobo-method, as.lobo, lobo-class, lobo, plot,lobo-method, summary,lobo-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-met subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

---

subset,met-method *Subset a met Object*

Description

This function is somewhat analogous to subset.data.frame.

Usage

```
## S4 method for signature 'met'
subset(x, subset, ...)
```

Arguments

x	An object inheriting from met-class.
subset	An expression indicating how to subset x.
...	ignored.

Value

A new met object.

Author(s)

Dan Kelley

See Also

Other things related to met data: [[,met-method, [[<-,met-method, as.met, download.met, met-class, met, plot,met-method, read.met, summary,met-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-met subset,lobo-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(met)
# Few days surrounding Hurricane Juan
plot(subset(met, time > as.POSIXct("2003-09-27", tz="UTC")))
```

`subset,oce-method` *Subset an oce Object*

Description

This is a basic class for general oce objects. It has specialised versions for most sub-classes, e.g. subset,ctd-method for ctd objects.

Usage

```
## S4 method for signature 'oce'
subset(x, subset, ...)
```

Arguments

x	an oce object.
subset	a logical expression indicating how to take the subset; the form depends on the sub-class.
...	optional arguments, used in some specialized methods (e.g. subset,section-method).

Value

An oce object.

See Also

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-met subset,lobo-method, subset,met-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(ctd)
# Select just the top 10 metres (pressure less than 10 dbar)
top10 <- subset(ctd, pressure < 10)
par(mfrow=c(1, 2))
plotProfile(ctd)
plotProfile(top10)
```

subset,odf-method *Subset an ODF object*

Description

This function is somewhat analogous to subset.data.frame.

Usage

```
## S4 method for signature 'odf'
subset(x, subset, ...)
```

Arguments

x	an odf object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Value

A new odf object.

Author(s)

Dan Kelley

See Also

Other things related to odf data: ODF2oce, ODFNames2oceNames, [[,odf-method, [[<-,odf-method, odf-class, plot,odf-method, read.ctd.odf, read.odf, summary,odf-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-method, subset,lobo-method, subset,met-method, subset,oce-method, subset,rsk-method, subset,sealevel-method, subset,section-method, subset,topo-method

subset,rsk-method *Subset a Rsk Object*

Description

Subset a rsk object. This function is somewhat analogous to subset.data.frame, but subsetting is only permitted by time.

Usage

```
## S4 method for signature 'rsk'
subset(x, subset, ...)
```

Arguments

x	a rsk object, i.e. inheriting from rsk-class.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Value

A new rsk object.

Author(s)

Dan Kelley

See Also

Other things related to rsk data: [[,rsk-method, [[<-,rsk-method, as.rsk, plot,rsk-method, read.rsk, rsk-class, rskPatm, rskToc, rsk, summary,rsk-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-met subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,sealevel-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(rsk)
plot(rsk)
plot(subset(rsk, time < mean(range(rsk[['time']]))))
```

---

subset,sealevel-method *Subset a Sealevel Object*

Description

This function is somewhat analogous to subset.data.frame, but subsetting is only permitted by time.

Usage

```
## S4 method for signature 'sealevel'
subset(x, subset, ...)
```

Arguments

x	A sealevel object, i.e. one inheriting from sealevel-class.
subset	a condition to be applied to the data portion of x.
...	ignored.

Value

A new sealevel object.

Author(s)

Dan Kelley

See Also

Other things related to sealevel data: [[,sealevel-method, [[<-,sealevel-method, as.sealevel, plot,sealevel-method, read.sealevel, sealevel-class, sealevelTuktoyaktuk, sealevel, summary,sealevel-method

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-met subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,section-method, subset,topo-method

Examples

```
library(oce)
data(sealevel)
plot(sealevel)
plot(subset(sealevel, time < mean(range(sealevel[['time']]))))
```

---

subset,section-method *Subset a Section Object*

Description

Return a subset of a section object.

Usage

```
## S4 method for signature 'section'
subset(x, subset, ...)
```

Arguments

x	A section object, i.e. one inheriting from section-class.
subset	an optional indication of either the stations to be kept, or the data to be kept within the stations. See "Details".
...	optional arguments, of which only the first is examined. The only possibility is that this argument be named indices. See "Details".

Details

This function is used to subset data within the stations of a section, or to choose a subset of the stations themselves. The first case is handled with the subset argument, while the second is handled if ... contains a vector named indices. Either subset or indices must be provided, but not both.

In the "subset" method, subset indicates either stations to be kept, or data to be kept within the stations.

The first step in processing is to check for the presence of certain key words in the subset expression. If distance is present, then stations are selected according to a condition on the distance (in km) from the first station to the given station (Example 1). If either longitude or latitude is given, then stations are selected according to the stated condition (Example 2). If stationId is present, then selection is in terms of the station ID (not the sequence number) is used (Example 3). In all of these cases, stations are either selected in their entirety or dropped in their entirety.

If none of these keywords is present, then the subset expression is evaluated in the context of the data slot of each of the CTD stations stored within x. (Note that this slot does not normally contain any of the keywords that are listed in the previous paragraph; it does, then odd results may occur.) Each station is examined in turn, with subset being evaluated individually in each. The evaluation produces a logical vector. If that vector has length 1 (Examples 4 and 5) then the station is retained or discarded, accordingly. If the vector is longer, then the logical vector is used as a sieve to subsample that individual CTD profile.

In the "indices" method, stations are selected using indices, which may be a vector of integers that indicate sequence number, or a logical vector, again indicating which stations to keep.

Value

A section-class object.

Author(s)

Dan Kelley

See Also

Other functions that subset oce objects: subset,adp-method, subset,adv-method, subset,amsr-method, subset,argo-method, subset,cm-method, subset,coastline-method, subset,ctd-method, subset,echosounder-met subset,lobo-method, subset,met-method, subset,oce-method, subset,odf-method, subset,rsk-method, subset,sealevel-method, subset,topo-method

Other things related to section data: [[,section-method, [[<-,section-method, as.section, handleFlags,section-method, plot,section-method, read.section, section-class, sectionAddStation, sectionGrid, sectionSmooth, sectionSort, section, summary,section-method

Examples

```
library(oce)
data(section)

# 1. Stations within 500 km of the first station
starting <- subset(section, distance < 500)

# 2. Stations east of 50W
east <- subset(section, longitude > (-50))

# 3. Gulf Stream
GS <- subset(section, 109 <= stationId & stationId <= 129)

# 4. Only stations with more than 5 pressure levels
long <- subset(section, length(pressure) > 5)

# 5. Only stations that have some data in top 50 dbar
surfacing <- subset(section, min(pressure) < 50)

# 6. Similar to #4, but done in more detailed way
long <- subset(section,
   indices=unlist(lapply(section[["station"]],
                  function(s)
                    5 < length(s[["pressure"]]))))
```

`subset,topo-method` *Subset a Topo Object*

Description

This function is somewhat analogous to `subset.data.frame`. Subsetting can be by `time` or `distance`, but these may not be combined; use a sequence of calls to subset by both.

Usage

```
## S4 method for signature 'topo'
subset(x, subset, ...)
```

Arguments

`x`	A topo object, i.e. inheriting from `topo-class`.
`subset`	A condition to be applied to the `data` portion of `x`. See 'Details'.
`...`	Ignored.

Value

A new `topo-class` object.

Author(s)

Dan Kelley

See Also

Other things related to topo data: `[[,topo-method`, `[[<-,topo-method`, `as.topo`, `download.topo`, `plot,topo-method`, `read.topo`, `summary,topo-method`, `topo-class`, `topoInterpolate`, `topoWorld`

Other functions that subset oce objects: `subset,adp-method`, `subset,adv-method`, `subset,amsr-method`, `subset,argo-method`, `subset,cm-method`, `subset,coastline-method`, `subset,ctd-method`, `subset,echosounder-met`
`subset,lobo-method`, `subset,met-method`, `subset,oce-method`, `subset,odf-method`, `subset,rsk-method`, `subset,sealevel-method`, `subset,section-method`

Examples

```
## northern hemisphere
library(oce)
data(topoWorld)
plot(subset(topoWorld, latitude > 0))
```

`subtractBottomVelocity` *Subtract Bottom Velocity from ADP*

Description

Subtracts bottom tracking velocities from an `"adp"` object. Works for all coordinate systems (`beam`, `xyz`, and `enu`).

Usage

```
subtractBottomVelocity(x, debug = getOption("oceDebug"))
```

Arguments

`x`	an object of class `"adp"`, which contains bottom tracking velocities.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Author(s)

Dan Kelley and Clark Richards

See Also

See `read.adp` for notes on functions relating to "adp" objects, and `adp-class` for notes on the ADP object class.

`summary,adp-method` *Summarize an ADP Object*

Description

Summarize data in an adp object.

Usage

```
## S4 method for signature 'adp'
summary(object, ...)
```

Arguments

`object`	an object of class "adp", usually, a result of a call to `read.oce`, `read.adp.rdi`, or `read.adp.nortek`.
`...`	further arguments passed to or from other methods.

Details

Pertinent summary information is presented.

Value

A matrix containing statistics of the elements of the `data` slot.

Author(s)

Dan Kelley

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu` `summary,adv-method` *Summarize an ADV object*

Description

Summarize data in an adv object.

Usage

```
## S4 method for signature 'adv'
summary(object, ...)
```

Arguments

`object`	an object of class `"adv"`, usually, a result of a call to `read.adv`.
`...`	further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOtherAdv`, `enuToOther`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnuAdv`, `xyzToEnu`

Examples

```
library(oce)
data(adv)
summary(adv)
```

`summary,amsr-method` *Summarize an AMSR Object*

Description

Although the data are stored in `raw` form, the summary presents results in physical units.

Usage

```
## S4 method for signature 'amsr'
summary(object, ...)
```

Arguments

`object`	The object to be summarized.
`...`	Ignored.

Author(s)

Dan Kelley

See Also

Other things related to amsr data: `[[<-,amsr-method`, `amsr-class`, `composite,amsr-method`, `download.amsr`, `plot,amsr-method`, `read.amsr`, `subset,amsr-method`

`summary,argo-method` *Summarize an Argo Object*

Description

Summarizes some of the data in an `argo` object.

Usage

```
## S4 method for signature 'argo'
summary(object, ...)
```

Arguments

`...`	Further arguments passed to or from other methods.
`object`	anobject of class `"argo"`, usually, a result of a call to `read.argo`.

Details

Pertinent summary information is presented.

Value

A matrix containing statistics of the elements of the `data` slot.

Author(s)

Dan Kelley

See Also

Other things related to argo data: `[[,argo-method`, `[[<-,argo-method`, `argo-class`, `argoGrid`, `argoNames2oceNames`, `argo`, `as.argo`, `handleFlags,argo-method`, `plot,argo-method`, `read.argo`, `subset,argo-method`

Examples

```
library(oce)
data(argo)
summary(argo)
```

---

summary,bremen-method *Summarize a Bremen Object*

Description

Pertinent summary information is presented, including the station name, sampling location, data ranges, etc.

Usage

```
## S4 method for signature 'bremen'
summary(object, ...)
```

Arguments

object	A bremen object, i.e. one inheriting from bremen-class. call to read.bremen.
...	Further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

Other things related to bremen data: [[,bremen-method, [[<-,bremen-method, bremen-class, plot,bremen-method, read.bremen

---

summary,cm-method *Summarize a CM Object*

Description

Summarizes some of the data in a cm object, presenting such information as the station name, sampling location, data ranges, etc.

Usage

```
## S4 method for signature 'cm'
summary(object, ...)
```

Arguments

`object`	A `cm` object, i.e. one inheriting from `cm-class`.
`...`	Further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

The documentation for `cm-class` explains the structure of `cm` objects, and also outlines the other functions dealing with them.

Other things related to `cm` data: `[[,cm-method`, `[[<-,cm-method`, `as.cm`, `cm-class`, `cm`, `plot,cm-method`, `read.cm`, `subset,cm-method`

`summary,coastline-method`

Summarize a Coastline Object

Description

Summarizes coastline length, bounding box, etc.

Usage

```
## S4 method for signature 'coastline'
summary(object, ...)
```

Arguments

`object`	an object of class `"coastline"`, usually, a result of a call to `read.coastline` or `read.oce`.
`...`	further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

Other things related to `coastline` data: `[[,coastline-method`, `[[<-,coastline-method`, `as.coastline`, `coastline-class`, `coastlineBest`, `coastlineCut`, `coastlineWorld`, `download.coastline`, `plot,coastline-method`, `read.coastline.openstreetmap`, `read.coastline.shapefile`, `subset,coastline-method` summary,ctd-method *Summarize a CTD Object*

Description

Summarizes some of the data in a ctd object, presenting such information as the station name, sampling location, data ranges, etc. If the object was read from a .cnv file or a .rsk file, then the OriginalName column for the data summary will contain the original names of data within the source file.

Usage

```
## S4 method for signature 'ctd'
summary(object, ...)
```

Arguments

object	A ctd object, i.e. one inheriting from ctd-class.
...	Further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
library(oce)
data(ctd)
summary(ctd)
```

`summary,echosounder-method` *Summarize an Echosounder Object*

Description

Summarizes some of the data in an `echosounder` object.

Usage

```
## S4 method for signature 'echosounder'
summary(object, ...)
```

Arguments

`object`	an object of class `"echosounder"`, usually, a result of a call to `read.echosounder`, `read.oce`, or `as.echosounder`.
`...`	further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

Other things related to `echosounder` data: `[[,echosounder-method`, `[[<-,echosounder-method`, `as.echosounder`, `echosounder-class`, `echosounder`, `findBottom`, `plot,echosounder-method`, `read.echosounder`, `subset,echosounder-method`

`summary,gps-method` *Summarize a GPS Object*

Description

Summarize a gps object, i.e. one inheriting from `gps-class`.

Usage

```
## S4 method for signature 'gps'
summary(object, ...)
```

Arguments

`object`	an object of class `"gps"`
`...`	further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

Other things related to gps data: [[,gps-method, [[<-,gps-method, as.gps, gps-class, plot,gps-method, read.gps

---

summary,ladp-method *Summarize an ladp object*

Description

Pertinent summary information is presented, including the station name, sampling location, data ranges, etc.

Usage

```
## S4 method for signature 'ladp'
summary(object, ...)
```

Arguments

object	An object of class "ladp", usually, a result of a call to as.ladp.
...	Further arguments passed to or from other methods.

Value

A matrix containing statistics of the elements of the data slot.

Author(s)

Dan Kelley

See Also

Other things related to ladp data: [[,ladp-method, as.ladp, ladp-class, plot,ladp-method

summary,landsat-method

Summarize a landsat Object

Description

Provides a summary of a some information about an object of landsat-class.

Usage

```
## S4 method for signature 'landsat'
summary(object, ...)
```

Arguments

object	An object of landsat-class, usually a result of a call to read.landsat.
...	Ignored.

Author(s)

Dan Kelley

See Also

Other things related to landsat data: [[,landsat-method, landsat-class, landsatAdd, landsatTrim, landsat, plot,landsat-method, read.landsat

summary,lisst-method

Summarize a LISST Object

Description

Summarizes some of the data in a lisst object, presenting such information as the station name, sampling location, data ranges, etc.

Usage

```
## S4 method for signature 'lisst'
summary(object, ...)
```

Arguments

object	An object of class lisst, usually, a result of a call to read.lisst or as.lisst.
...	Ignored.

Author(s)

Dan Kelley

See Also

Other things related to `lisst` data: `[[,lisst-method`, `[[<-,lisst-method`, `as.lisst`, `lisst-class`, `plot,lisst-method`, `read.lisst`

Examples

```
library(oce)
data(lisst)
summary(lisst)
```

---

summary,lobo-method *Summarize a LOBO Object*

Description

Pertinent summary information is presented, including the sampling interval, data ranges, etc.

Usage

```
## S4 method for signature 'lobo'
summary(object, ...)
```

Arguments

`object`	an object of class `"lobo"`, usually, a result of a call to `read.lobo` or `read.oce`.
`...`	further arguments passed to or from other methods.

Value

A matrix containing statistics of the elements of the `data` slot.

Author(s)

Dan Kelley

References

http://lobo.satlantic.com http://www.mbari.org/lobo/

See Also

The documentation for lobo-class explains the structure of LOBO objects, and also outlines the other functions dealing with them.

Other things related to lobo data: [[,lobo-method, [[<-,lobo-method, as.lobo, lobo-class, lobo, plot,lobo-method, subset,lobo-method

Examples

```
library(oce)
data(lobo)
summary(lobo)
```

---

summary,met-method *Summarize a met Object*

Description

Pertinent summary information is presented, including the sampling location, data ranges, etc.

Usage

```
## S4 method for signature 'met'
summary(object, ...)
```

Arguments

object	A met object, i.e. one inheriting from met-class.
...	further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

Other things related to met data: [[,met-method, [[<-,met-method, as.met, download.met, met-class, met, plot,met-method, read.met, subset,met-method

summary,oce-method *Summarize an oce Object*

Description

Provide a textual summary of some pertinent aspects of the object, including selected components of its `metadata` slot, statistical and dimensional information on the entries in the `data` slot, and a listing of the contents of its `processingLog` slot. The details depend on the class of the object, especially for the `metadata` slot, so it can help to consult the specialized documentation, e.g. `summary,ctd-method` for CTD objects (i.e. objects inheriting from `ctd-class`.) It is important to note that this is not a good way to learn the details of the object contents. Instead, for an object named `object`, say, one might use `str(object)` to learn about all the contents, or `str(object[["metadata"]])` to learn about the `metadata`, etc.

Usage

```
## S4 method for signature 'oce'
summary(object, ...)
```

Arguments

`object`	The object to be summarized.
`...`	Extra arguments (ignored)

Examples

```
o <- new("oce")
summary(o)
```

summary,odf-method *Summarize an ODF Object*

Description

Pertinent summary information is presented, including the station name, sampling location, data ranges, etc.

Usage

```
## S4 method for signature 'odf'
summary(object, ...)
```

Arguments

`object`	an object of class `"odf"`, usually, a result of a call to `read.odf` or `read.oce`.
`...`	further arguments passed to or from other methods.

Value

A matrix containing statistics of the elements of the data slot.

Author(s)

Dan Kelley

See Also

Other things related to odf data: ODF2oce, ODFNames2oceNames, [[,odf-method, [[<-,odf-method, odf-class, plot,odf-method, read.ctd.odf, read.odf, subset,odf-method

---

summary,rsk-method *Summarize a Rsk Object*

Description

Summarizes some of the data in a rsk object, presenting such information as the station name, sampling location, data ranges, etc.

Usage

```
## S4 method for signature 'rsk'
summary(object, ...)
```

Arguments

object	An object of class "rsk", usually, a result of a call to read.rsk, read.oce, or as.rsk.
...	Further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

The documentation for rsk-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Other things related to rsk data: [[,rsk-method, [[<-,rsk-method, as.rsk, plot,rsk-method, read.rsk, rsk-class, rskPatm, rskToc, rsk, subset,rsk-method

Examples

```
library(oce)
data(rsk)
summary(rsk)
```

`summary,satellite-method` *Summarize a satellite object*

Description

Summarize a satellite object

Usage

```
## S4 method for signature 'satellite'
summary(object, ...)
```

Arguments

`object`	The object to be summarized.
`...`	Ignored.

Author(s)

Dan Kelley

See Also

Other things related to satellite data: `g1sst-class`, `plot,satellite-method`, `read.g1sst`, `satellite-class`

`summary,sealevel-method` *Summarize a Sealevel Object*

Description

Summarizes some of the data in a sealevel object.

Usage

```
## S4 method for signature 'sealevel'
summary(object, ...)
```

Arguments

`object`	A `sealevel` object, i.e. one inheriting from `sealevel-class`.
`...`	further arguments passed to or from other methods.

Value

A matrix containing statistics of the elements of the data slot.

Author(s)

Dan Kelley

See Also

Other things related to sealevel data: [[,sealevel-method, [[<-,sealevel-method, as.sealevel, plot,sealevel-method, read.sealevel, sealevel-class, sealevelTuktoyaktuk, sealevel, subset,sealevel-method

Examples

```
library(oce)
data(sealevel)
summary(sealevel)
```

---

summary,section-method

Summarize a Section Object

Description

Pertinent summary information is presented, including station locations, distance along track, etc.

Usage

```
## S4 method for signature 'section'
summary(object, ...)
```

Arguments

object	An object of class "section", usually, a result of a call to read.section, read.oce, or as.section.
...	Further arguments passed to or from other methods.

Value

NULL

Author(s)

Dan Kelley

See Also

Other things related to section data: [[,section-method, [[<-,section-method, as.section, handleFlags,section-method, plot,section-method, read.section, section-class, sectionAddStation, sectionGrid, sectionSmooth, sectionSort, section, subset,section-method

Examples

```
library(oce)
data(section)
summary(section)
```

---

summary,tidem-method *Summarize a Tidem Object*

Description

By default, all fitted constituents are plotted, but it is quite useful to set e.g. p=0.05 To see just those constituents that are significant at the 5 percent level. Note that the p values are estimated as the average of the p values for the sine and cosine components at a given frequency.

Usage

```
## S4 method for signature 'tidem'
summary(object, p, constituent, ...)
```

Arguments

object	an object of class "tidem", usually, a result of a call to tidem.
p	optional value of the maximum p value for the display of an individual coefficient. If not given, all coefficients are shown.
constituent	optional name of constituent on which to focus.
...	further arguments passed to or from other methods.

Value

NULL

Author(s)

Dan Kelley

See Also

Other things related to tidem data: [[,tidem-method, [[<-,tidem-method, plot,tidem-method, predict.tidem, tidedata, tidem-class, tidemAstron, tidemVuf, tidem

Examples

```
## Not run:
library(oce)
data(sealevel)
tide <- tidem(sealevel)
summary(tide)

## End(Not run)
```

---

summary,topo-method *Summarize A Topo Object*

Description

Pertinent summary information is presented, including the longitude and latitude range, and the range of elevation.

Usage

```
## S4 method for signature 'topo'
summary(object, ...)
```

Arguments

`object`	A `topo` object, i.e. inheriting from `topo-class`.
`...`	Further arguments passed to or from other methods.

Value

A matrix containing statistics of the elements of the `data` slot.

Author(s)

Dan Kelley

See Also

Other things related to `topo` data: `[[,topo-method`, `[[<-,topo-method`, `as.topo`, `download.topo`, `plot,topo-method`, `read.topo`, `subset,topo-method`, `topo-class`, `topoInterpolate`, `topoWorld`

Examples

```
library(oce)
data(topoWorld)
summary(topoWorld)
```

summary,windrose-method *Summarize a* windrose *object*

Description

Summarizes some of the data in a windrose object.

Usage

```
## S4 method for signature 'windrose'
summary(object, ...)
```

Arguments

object	An windrose object, i.e. inheriting from windrose-class.
...	Further arguments passed to or from other methods.

Author(s)

Dan Kelley

See Also

The documentation for windrose-class explains the structure of windrose objects, and also outlines the other functions dealing with them.

Other things related to windrose data: [[,windrose-method, [[<-,windrose-method, as.windrose, plot,windrose-method, windrose-class

sunAngle *Solar Angle as Function of Space and Time*

Description

Solar angle as function of space and time.

Usage

```
sunAngle(t, longitude = 0, latitude = 0, useRefraction = FALSE)
```

Arguments

t	time, a POSIXt object (converted to timezone "UTC", if it is not already in that timezone), or a numeric value that corresponds to such a time.
longitude	observer longitude in degrees east
latitude	observer latitude in degrees north
useRefraction	boolean, set to TRUE to apply a correction for atmospheric refraction

Details

Based on NASA-provided Fortran program, in turn (according to comments in the code) based on "The Astronomical Almanac".

Value

A list containing the following.

`time`	time
`azimuth`	azimuth, in degrees eastward of north, from 0 to 360. (See diagram below.)
`altitude`	altitude, in degrees above the horizon, ranging from -90 to 90. (See diagram below.)
`diameter`	solar diameter, in degrees
`distance`	distance to sun, in astronomical units

Author(s)

Dan Kelley

References

Based on Fortran code retrieved from ftp://climate1.gsfc.nasa.gov/wiscombe/Solar_Rad/SunAngles/sunae.f on 2009-11-1. Comments in that code list as references:

Michalsky, J., 1988: The Astronomical Almanac's algorithm for approximate solar position (1950-2050), Solar Energy 40, 227-235

The Astronomical Almanac, U.S. Gov't Printing Office, Washington, D.C. (published every year).

The code comments suggest that the appendix in Michalsky (1988) contains errors, and declares the use of the following formulae in the 1995 version the Almanac:

- p. A12: approximation to sunrise/set times;
- p. B61: solar altitude (AKA elevation) and azimuth;
- p. B62: refraction correction;
- p. C24: mean longitude, mean anomaly, ecliptic longitude, obliquity of ecliptic, right ascension, declination, Earth-Sun distance, angular diameter of Sun;
- p. L2: Greenwich mean sidereal time (ignoring T^2, T^3 terms).

The code lists authors as Dr. Joe Michalsky and Dr. Lee Harrison (State University of New York), with modifications by Dr. Warren Wiscombe (NASA Goddard Space Flight Center).

See Also

The equivalent function for the moon is `moonAngle`.

Other things related to astronomy: `eclipticalToEquatorial`, `equatorialToLocalHorizontal`, `julianCenturyAnomaly`, `julianDay`, `moonAngle`, `siderealTime`

Examples

```
rise <- as.POSIXct("2011-03-03 06:49:00", tz="UTC") + 4*3600
set <- as.POSIXct("2011-03-03 18:04:00", tz="UTC") + 4*3600
mismatch <- function(lonlat)
{
   sunAngle(rise, lonlat[1], lonlat[2])$altitude^2 + sunAngle(set, lonlat[1], lonlat[2])$altitude^2
}
result <- optim(c(1,1), mismatch)
lon.hfx <- (-63.55274)
lat.hfx <- 44.65
dist <- geodDist(result$par[1], result$par[2], lon.hfx, lat.hfx)
cat(sprintf("Infer Halifax latitude %.2f and longitude %.2f; distance mismatch %.0f km",
            result$par[2], result$par[1], dist))
```

`swAbsoluteSalinity` *Seawater absolute salinity, in GSW formulation*

Description

Compute seawater absolute salinity, according to the GSW/TEOS-10 formulation.

Usage

```
swAbsoluteSalinity(salinity, pressure = NULL, longitude = NULL,
  latitude = NULL)
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an oce object (in which case `salinity`, etc. are inferred from the object).
`pressure`	pressure in dbar.
`longitude`	longitude of observation.
`latitude`	latitude of observation.

Details

The absolute salinity is calculated using the GSW function `gsw_SA_from_SP`. Typically, this is a fraction of a unit higher than practical salinity as defined in the UNESCO formulae.

Value

Absolute Salinity in g/kg.

Author(s)

Dan Kelley

References

McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

See Also

The related TEOS-10 quantity "conservative temperature" may be computed with `swConservativeTemperature`. For a ctd object, absolute salinity may also be recovered by indexing as e.g. `ctd[["absoluteSalinity"]]` or `ctd[["SA"]]`.

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
## Not run:
sa <- swAbsoluteSalinity(35.5, 300, 260, 16)
stopifnot(abs(35.671358392019094 - sa) < 00.000000000000010)

## End(Not run)
```

`swAlpha` *Seawater thermal expansion coefficient*

Description

Compute α, the thermal expansion coefficient for seawater.

Usage

```
swAlpha(salinity, temperature = NULL, pressure = 0, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an `oce` object (in which case `salinity`, etc. are inferred from the object).
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale; see "Temperature units" in the documentation for `swRho`.
`pressure`	pressure [dbar]
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` or `"gsw"`.

Value

Value in 1/degC.

Author(s)

Dan Kelley

References

The eos="unesco" formulae are based on the UNESCO equation of state, but are formulated empirically by Trevor J. McDougall, 1987, Neutral Surfaces, Journal of Physical Oceanography, volume 17, pages 1950-1964. The eos="gsw" formulae come from GSW; see references in the swRho documentation.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

---

swAlphaOverBeta — *Ratio of seawater thermal expansion coefficient to haline contraction coefficient*

Description

Compute α/β using McDougall's (1987) algorithm.

Usage

```
swAlphaOverBeta(salinity, temperature = NULL, pressure = NULL,
  longitude = NULL, latitude = NULL, eos = getOption("oceEOS", default =
  "gsw"))
```

Arguments

salinity	either practical salinity (in which case temperature and pressure must be provided) **or** an oce object (in which case salinity, etc. are inferred from the object).
temperature	*in-situ* temperature [°C]
pressure	pressure [dbar]
longitude	longitude of observation (only used if eos="gsw"; see 'Details').
latitude	latitude of observation (only used if eos="gsw"; see 'Details').
eos	equation of state, either "unesco" or "gsw".

Value

Value in psu/°C.

Author(s)

Dan Kelley

References

The `eos="unesco"` formulae are based on the UNESCO equation of state, but are formulated empirically by Trevor J. McDougall, 1987, Neutral Surfaces, Journal of Physical Oceanography, volume 17, pages 1950-1964. The `eos="gsw"` formulae come from GSW; see references in the `swRho` documentation.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
swAlphaOverBeta(40, 10, 4000, eos="unesco") # 0.3476
```

`swBeta` *Seawater haline contraction coefficient*

Description

Compute β, the haline contraction coefficient for seawater.

Usage

```
swBeta(salinity, temperature = NULL, pressure = 0, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an `oce` object (in which case `salinity`, etc. are inferred from the object).
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale; see "Temperature units" in the documentation for `swRho`.
`pressure`	seawater pressure [dbar]

`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` or `"gsw"`.

Value

Value in 1/psu.

Author(s)

Dan Kelley

References

The `eos="unesco"` formulae are based on the UNESCO equation of state, but are formulaed empirically by Trevor J. McDougall, 1987, Neutral Surfaces, Journal of Physical Oceanography, volume 17, pages 1950-1964. The `eos="gsw"` formulae come from GSW; see references in the `swRho` documentation.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

`swConservativeTemperature`

Seawater conservative temperature, in GSW formulation

Description

Compute seawater Conservative Temperature, according to the GSW/TEOS-10 formulation.

Usage

```
swConservativeTemperature(salinity, temperature = NULL, pressure = NULL,
  longitude = NULL, latitude = NULL)
```

Arguments

salinity	either practical salinity (in which case temperature and pressure must be provided) **or** an oce object (in which case salinity, etc. are inferred from the object).
temperature	*in-situ* temperature [°C], defined on the ITS-90 scale; see "Temperature units" in the documentation for swRho.
pressure	pressure [dbar]
longitude	longitude of observation.
latitude	latitude of observation.

Details

If the first argument is an oce object, then values for salinity, etc., are extracted from it, and used for the calculation, and the corresponding arguments to the present function are ignored.

The conservative temperature is calculated using the TEOS-10 function gsw_CT_from_t from the gsw package.

Value

Conservative temperature in degrees Celcius.

Author(s)

Dan Kelley

References

McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

See Also

The related TEOS-10 quantity "absolute salinity" may be computed with swAbsoluteSalinity. For a ctd object, conservative temperature may also be recovered by indexing as e.g. ctd[["conservativeTemperature"]] or ctd[["CT"]].

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
swConservativeTemperature(35,10,1000,188,4) # 9.86883
```

swCSTp — *Electrical conductivity ratio from salinity, temperature and pressure*

Description

Compute electrical conductivity ratio based on salinity, temperature, and pressure (relative to the conductivity of seawater with salinity=35, temperature68=15, and pressure=0).

Usage

```
swCSTp(salinity, temperature = 15, pressure = 0, eos = getOption("oceEOS",
  default = "gsw"))
```

Arguments

`salinity`	practical salinity, or a CTD object (in which case its temperature and pressure are used, and the next two arguments are ignored)
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale; see the examples, as well as the "Temperature units" section in the documentation for `swRho`.
`pressure`	pressure [dbar]
`eos`	equation of state, either `"unesco"` or `"gsw"`.

Details

If `eos="unesco"`, the calculation is done by a bisection root search on the UNESCO formula relating salinity to conductivity, temperature, and pressure (see `swSCTp`). If it is `"gsw"` then the Gibbs-SeaWater formulation is used, via `gsw_C_from_SP`.

Value

Conductivity ratio [unitless], i.e. the ratio of conductivity to the conductivity at salinity=35, temperature=15 (IPTS-68 scale) and pressure=0, which has numerical value 42.9140 mS/cm = 4.29140 S/m (see Culkin and Smith, 1980, in the regression result cited at the bottom of the left-hand column on page 23).

Author(s)

Dan Kelley

References

1. Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp.

2. Culkin, F., and Norman D. Smith, 1980. Determination of the concentration of potassium chloride solution having the same electrical conductivity, at 15 C and infinite frequency, as standard seawater of salinity 35.0000 ppt (Chlorinity 19.37394 ppt). *IEEE Journal of Oceanic Engineering*, **5**, pp 22-23.

See Also

For thermal (as opposed to electrical) conductivity, see swThermalConductivity. For computation of salinity from electrical conductivity, see swSCTp.

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
expect_equal(1, swCSTp(35, T90fromT68(15), 0, eos="unesco")) # by definition of cond. ratio
expect_equal(1, swCSTp(34.25045, T90fromT68(15), 2000, eos="unesco"), tolerance=1e-7)
expect_equal(1, swCSTp(34.25045, T90fromT68(15), 2000, eos="gsw"), tolerance=1e-7)
```

---

swDepth *Water depth*

Description

Compute depth below the surface (i.e. a positive number within the water column) based on pressure and latitude. (Use swZ to get the vertical coordinate, which is negative within the water column.)

Usage

```
swDepth(pressure, latitude = 45, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

pressure	either pressure [dbar], in which case lat must also be given, or a ctd object, in which case lat will be inferred from the object.
latitude	Latitude in °N or radians north of the equator.
eos	indication of formulation to be used, either "unesco" or "gsw".

Details

If eos="unesco" then depth is calculated from pressure using Saunders and Fofonoff's method, with the formula refitted for 1980 UNESCO equation of state [1]. If eos="gsw", then gsw_z_from_p from the gsw package [2,3] is used.

Value

Depth below the ocean surface, in metres.

Author(s)

Dan Kelley

References

1. Unesco 1983. Algorithms for computation of fundamental properties of seawater, 1983. *Unesco Tech. Pap. in Mar. Sci.*, No. 44, 53 pp.

2. IOC, SCOR, and IAPSO (2010). The international thermodynamic equation of seawater-2010: Calculation and use of thermodynamic properties. Technical Report 56, Intergovernmental Oceanographic Commission, Manuals and Guide.

3. McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
d <- swDepth(10, 45)
```

`swDynamicHeight` *Dynamic height of seawater profile*

Description

Compute the dynamic height of a column of seawater.

Usage

```
swDynamicHeight(x, referencePressure = 2000, subdivisions = 500,
  rel.tol = .Machine$double.eps^0.25, eos = getOption("oceEOS", default =
  "gsw"))
```

Arguments

`x` a `section` object, **or** a `ctd` object.

`referencePressure` reference pressure [dbar]. If this exceeds the highest pressure supplied to `swDynamicHeight`, then that highest pressure is used, instead of the supplied value of `referencePressure`.

subdivisions number of subdivisions for call to integrate. (The default value is considerably larger than the default for integrate, because otherwise some test profiles failed to integrate.

rel.tol absolute tolerance for call to integrate. Note that this call is made in scaled coordinates, i.e. pressure is divided by its maximum value, and dz/dp is also divided by its maximum.

eos equation of state, either "unesco" or "gsw".

Details

If the first argument is a `section`, then dynamic height is calculated for each station within a section, and returns a list containing distance along the section along with dynamic height.

If the first argument is a `ctd`, then this returns just a single value, the dynamic height.

If `eos="unesco"`, processing is as follows. First, a piecewise-linear model of the density variation with pressure is developed using approxfun. (The option `rule=2` is used to extrapolate the uppermost density up to the surface, preventing a possible a bias for bottle data, in which the first depth may be a few metres below the surface.) A second function is constructed as the density of water with salinity 35PSU, temperature of 0°C, and pressure as in the `ctd`. The difference of the reciprocals of these densities, is then integrated with integrate with pressure limits `0` to `referencePressure`. (For improved numerical results, the variables are scaled before the integration, making both independent and dependent variables be of order one.)

If `eos="gsw"`, gsw_geo_strf_dyn_height is used to calculate a result in m^2/s^2, and this is divided by $9.7963 m/s^2$. If pressures are out of order, the data are sorted. If any pressure is repeated, only the first level is used. If there are under 4 remaining distinct pressures, `NA` is returned, with a warning.

Value

In the first form, a list containing `distance`, the distance [km] from the first station in the section and `height`, the dynamic height [m].

In the second form, a single value, containing the dynamic height [m].

Author(s)

Dan Kelley

References

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
## Not run:
library(oce)
data(section)

# Dynamic height and geostrophy
par(mfcol=c(2,2))
par(mar=c(4.5,4.5,2,1))

# Left-hand column: whole section
# (The smoothing lowers Gulf Stream speed greatly)
westToEast <- subset(section, 1<=stationId&stationId<=123)
dh <- swDynamicHeight(westToEast)
plot(dh$distance, dh$height, type='p', xlab="", ylab="dyn. height [m]")
ok <- !is.na(dh$height)
smu <- supsmu(dh$distance, dh$height)
lines(smu, col="blue")
f <- coriolis(section[["station", 1]][["latitude"]])
g <- gravity(section[["station", 1]][["latitude"]])
v <- diff(smu$y)/diff(smu$x) * g / f / 1e3 # 1e3 converts to m
plot(smu$x[-1], v, type='l', col="blue", xlab="distance [km]", ylab="velocity [m/s]")

# right-hand column: gulf stream region, unsmoothed
gs <- subset(section, 102<=stationId&stationId<=124)
dh.gs <- swDynamicHeight(gs)
plot(dh.gs$distance, dh.gs$height, type='b', xlab="", ylab="dyn. height [m]")
v <- diff(dh.gs$height)/diff(dh.gs$distance) * g / f / 1e3
plot(dh.gs$distance[-1], v, type='l', col="blue",
  xlab="distance [km]", ylab="velocity [m/s]")

## End(Not run)
```

---

swLapseRate *Seawater lapse rate*

Description

Compute adiabatic lapse rate

Usage

```
swLapseRate(salinity, temperature = NULL, pressure = NULL,
  longitude = NULL, latitude = NULL, eos = getOption("oceEOS", default =
  "gsw"))
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale; see "Temperature units" in the documentation for `swRho`.
`pressure`	pressure [dbar]
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

If `eos="unesco"`, the density is calculated using the UNESCO equation of state for seawater [1,2], and if `eos="gsw"`, the GSW formulation [3,4] is used.

Value

Lapse rate [degC/m].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp. (Section 7, pages 38-40)

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
lr <- swLapseRate(40, 40, 10000) # 3.255976e-4
```

swN2 *Squared buoyancy frequency for seawater*

Description

Compute N^2, the square of the buoyancy frequency for a seawater profile.

Usage

```
swN2(pressure, sigmaTheta = NULL, derivs, df, eos = getOption("oceEOS",
  default = "gsw"), ...)
```

Arguments

pressure	either pressure [dbar] (in which case sigmaTheta must be provided) **or** an object of class ctd object (in which case sigmaTheta is inferred from the object.
sigmaTheta	Surface-referenced potential density minus 1000 [kg/m^3]
derivs	optional argument to control how the derivative $d\sigma_\theta/dp$ is calculated. This may be a character string or a function of two arguments. See "Details".
df	argument passed to smooth.spline if this function is used for smoothing; set to NA to prevent smoothing.
eos	equation of state, either "unesco" or "gsw".
...	additional argument, passed to smooth.spline, in the case that derivs="smoothing". See "Details".

Details

Smoothing is often useful prior to computing buoyancy frequency, and so this may optionally be done with smooth.spline, unless df=NA, in which case raw data are used. If df is not provided, a possibly reasonable value computed from an analysis of the profile, based on the number of pressure levels.

If eos="gsw", then the first argument must be a ctd object, and processing is done with gsw_Nsquared, based on extracted values of Absolute Salinity and Conservative Temperature (possibly smoothed, depending on df).

If eos="unesco", then the processing is as follows. The core of the method involves differentiating potential density (referenced to median pressure) with respect to pressure, and the derivs argument is used to control how this is done, as follows.

- if derivs is not supplied, the action is as though it were given as the string "smoothing"
- if derivs equals "simple", then the derivative of density with respect to pressure is calculated as the ratio of first-order derivatives of density and pressure, each calculated using diff. (A zero is appended at the top level.)

- if derivs equals "smoothing", then the processing depends on the number of data in the profile, and on whether df is given as an optional argument. When the number of points exceeds 4, and when df exceeds 1, smooth.spline is used to calculate smoothing spline representation the variation of density as a function of pressure, and derivatives are extracted from the spline using predict. Otherwise, density is smoothed using smooth, and derivatives are calculated as with the "simple" method.
- if derivs is a function taking two arguments (first pressure, then density) then that function is called directly to calculate the derivative, and no smoothing is done before or after that call.

For deep-sea work, the eos="gsw" option is the best scheme, because it uses derivatives of density computed with *local* reference pressure.

For precise work, it makes sense to skip swN2 entirely, choosing whether, what, and how to smooth based on an understanding of fundamental principles as well as data practicalities.

Value

Square of buoyancy frequency $[radian^2/s^2]$.

Author(s)

Dan Kelley

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
library(oce)
data(ctd)
# Illustrate difference between UNESCO and GSW
p <- ctd[["pressure"]]
ylim <- rev(range(p))
par(mfrow=c(1,3), mar=c(3, 3, 1, 1), mgp=c(2, 0.7, 0))
plot(ctd[["sigmaTheta"]], p, ylim=ylim, type='l', xlab=expression(sigma[theta]))
N2u <- swN2(ctd, eos="unesco")
N2g <- swN2(ctd, eos="gsw")
plot(N2u, p, ylim=ylim, xlab="N2 Unesco", ylab="p", type="l")
d <- 100 * (N2u - N2g) / N2g
plot(d, p, ylim=ylim, xlab="N2 UNESCO-GSW diff. [%]", ylab="p", type="l")
abline(v=0)
```

swPressure *Water pressure*

Description

Compute seawater pressure from depth by inverting `swDepth` using `uniroot`.

Usage

```
swPressure(depth, latitude = 45, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`depth`	distance below the surface in metres.
`latitude`	Latitude in °N or radians north of the equator.
`eos`	indication of formulation to be used, either `"unesco"` or `"gsw"`.

Details

If `eos="unesco"` this is done by numerical inversion of `swDepth` is done using `uniroot`. If `eos="gsw"`, it is done using `gsw_p_from_z` in the gsw package.

Value

Pressure in dbar.

Author(s)

Dan Kelley

References

Unesco 1983. Algorithms for computation of fundamental properties of seawater, 1983. *Unesco Tech. Pap. in Mar. Sci.*, No. 44, 53 pp.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
swPressure(9712.653, 30, eos="unesco") # 10000
swPressure(9712.653, 30, eos="gsw")    #  9998.863
```

`swRho` *Seawater density*

Description

Compute ρ, the *in-situ* density of seawater.

Usage

```
swRho(salinity, temperature = NULL, pressure = NULL, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an oce object, in which case `salinity`, `temperature` (in the ITS-90 scale; see next item), etc. are inferred from the object.
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting `eos="gsw"`), and is the value contained within `ctd` objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using `T68fromT90`.
`pressure`	pressure [dbar]
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

If `eos="unesco"`, the density is calculated using the UNESCO equation of state for seawater [1,2], and if `eos="gsw"`, the GSW formulation [3,4] is used.

Value

In-situ density [kg/m^3].

Temperature units

The UNESCO formulae are defined in terms of temperature measured on the IPTS-68 scale, whereas the replacement GSW formulae are based on the ITS-90 scale. Prior to the addition of GSW capabilities, the various `sw*` functions took temperature to be in IPTS-68 units. As GSW capabilities were added in early 2015, the assumed unit of `temperature` was taken to be ITS-90. This change means that old code has to be modified, by replacing e.g. `swRho(S, T, p)` with `swRho(S, T90fromT68(T), p)`. At typical oceanic values, the difference between the two scales is a few millidegrees.

Author(s)

Dan Kelley

References

1. Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

2. Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

3. IOC, SCOR, and IAPSO (2010). The international thermodynamic equation of seawater-2010: Calculation and use of thermodynamic properties. Technical Report 56, Intergovernmental Oceanographic Commission, Manuals and Guide.

4. McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

See Also

Related density routines include `swSigma0` (and equivalents at other pressure horizons), `swSigmaT`, and `swSigmaTheta`.

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
library(oce)
# The numbers in the comments are the check values listed in reference [1];
# note that temperature in that reference was on the T68 scale, but that
# the present function works with the ITS-90 scale, so a conversion
# is required.
swRho(35, T90fromT68(5),      0, eos="unesco") # 1027.67547
swRho(35, T90fromT68(5),  10000, eos="unesco") # 1069.48914
swRho(35, T90fromT68(25),     0, eos="unesco") # 1023.34306
swRho(35, T90fromT68(25), 10000, eos="unesco") # 1062.53817
```

`swRrho` *Density ratio*

Description

Compute density ratio

Usage

```
swRrho(ctd, sense = c("diffusive", "finger"), smoothingLength = 10, df,
  eos = getOption("oceEOS", default = "gsw"))
```

Arguments

- `ctd` — an object of class `ctd`
- `sense` — an indication of the sense of double diffusion under study and therefore of the definition of Rrho; see 'Details'
- `smoothingLength` — ignored if `df` supplied, but otherwise the latter is calculated as the number of data points, divided by the number within a depth interval of `smoothingLength` metres.
- `df` — if given, this is provided to `smooth.spline`.
- `eos` — equation of state, either `"unesco"` or `"gsw"`.

Details

This computes Rrho (density ratio) from a `ctd` object.

If `eos="unesco"`, this is done by calculating salinity and potential-temperature derivatives from smoothing splines whose properties are governed by `smoothingLength` or `df`. If `sense="diffusive"` the definition is $(beta * dS/dz)/(alpha * d(theta)/dz)$ and the reciprocal for `"finger"`.

If `eos="gsw"`, this is done by extracting absolute salinity and conservative temperature, smoothing with a smoothing spline as in the `"unesco"` case, and then calling `gsw_Turner_Rsubrho` on these smoothed fields. Since the gsw function works on mid-point pressures, `approx` is used to interpolate back to the original pressures.

If the default arguments are acceptable, `ctd[["Rrho"]]` may be used instead of `swRrho(ctd)`.

Value

Density ratio defined in either the `"diffusive"` or `"finger"` sense.

Author(s)

Dan Kelley and Chantelle Layton

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
library(oce)
data(ctd)
u <- swRrho(ctd, eos="unesco")
g <- swRrho(ctd, eos="gsw")
p <- ctd[["p"]]
plot(u, p, ylim=rev(range(p)), type='l', xlab=expression(R[rho]))
lines(g, p, lty=2, col='red')
legend("topright", lty=1:2, legend=c("unesco", "gsw"), col=c("black", "red"))
```

---

swSCTp *Salinity from electrical conductivity, temperature and pressure*

Description

Compute salinity based on electrical conductivity, temperature, and pressure.

Usage

```
swSCTp(conductivity, temperature = NULL, pressure = NULL, conductivityUnit,
  eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`conductivity`	a measure of conductivity (see also `conductivityUnit`) or an `oce` object holding hydrographic information. In the second case, all the other arguments to `swSCTp` are ignored.
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale; see "Temperature units" in the documentation for `swRho`.
`pressure`	pressure [dbar]
`conductivityUnit`	string indicating the unit used for conductivity. This may be `"ratio"` or `""` (meaning conductivity ratio), `"mS/cm"` or `"S/m"`. Note that the ratio mode assumes that measured conductivity has been divided by the standard conductivity of 4.2914 S/m.
`eos`	equation of state, either `"unesco"` or `"gsw"`.

Details

Calculate salinity from what is actually measured by a CTD, *i.e.* conductivity, *in-situ* temperature and pressure. Often this is done by the CTD processing software, but sometimes it is helpful to do this directly, *e.g.* when there is a concern about mismatches in sensor response times. If `eos="unesco"` then salinity is calculated using the UNESCO algorithm described by Fofonoff and Millard (1983); if it is `"gsw"` then the Gibbs-SeaWater formulation is used, via `gsw_SP_from_C`.

Value

Practical salinity.

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

See Also

For thermal (as opposed to electrical) conductivity, see swThermalConductivity. For computation of electrical conductivity from salinity, see swCSTp.

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
swSCTp(1, T90fromT68(15), 0, eos="unesco") # 35
swSCTp(1, T90fromT68(15), 0, eos="gsw")    # 35
```

---

swSigma *Seawater density anomaly*

Description

Compute σ_θ, the density of seawater, minus 1000 kg/m^3.

Usage

```
swSigma(salinity, temperature = NULL, pressure = NULL, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity` either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an oce object, in which case `salinity`, `temperature` (in the ITS-90 scale; see next item), etc. are inferred from the object.

`temperature` *in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting `eos="gsw"`), and is the value contained within `ctd` objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using `T68fromT90`.

`pressure` pressure [dbar]

`longitude` longitude of observation (only used if `eos="gsw"`; see 'Details').

`latitude` latitude of observation (only used if `eos="gsw"`; see 'Details').

`eos` equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Value

Density anomaly [kg/m^3], defined as `swRho` - 1000 kg/m^3.

Author(s)

Dan Kelley

References

See citations provided in the `swRho` documentation.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
library(oce)
swSigma(35, 13, 1000, longitude=300, latitude=30, eos="gsw") # 30.82374
swSigma(35, T90fromT68(13), 1000, eos="unesco")    # 30.8183
```

`swSigma0` *Seawater potential density anomaly referenced to surface pressure*

Description

Compute σ_θ, the potential density of seawater (minus 1000 kg/m^3), referenced to surface pressure.

Usage

```
swSigma0(salinity, temperature = NULL, pressure = NULL, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an oce object, in which case `salinity`, `temperature` (in the ITS-90 scale; see next item), etc. are inferred from the object.
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting `eos="gsw"`), and is the value contained within `ctd` objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using `T68fromT90`.
`pressure`	pressure [dbar]
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

Definition: $\sigma_0 = \sigma_\theta = \rho(S, \theta(S, t, p), 0$ - 1000 kg/m^3.

Value

Potential density anomaly [kg/m^3].

Author(s)

Dan Kelley

References

See citations provided in the `swRho` documentation.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

---

swSigma1 *Seawater potential density anomaly referenced to 1000db pressure*

Description

Compute σ_θ, the potential density of seawater (minus 1000 kg/m^3), referenced to 1000db pressure.

Usage

```
swSigma1(salinity, temperature = NULL, pressure = NULL, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

salinity	either practical salinity (in which case temperature and pressure must be provided) **or** an oce object, in which case salinity, temperature (in the ITS-90 scale; see next item), etc. are inferred from the object.
temperature	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting eos="gsw"), and is the value contained within ctd objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using T68fromT90.
pressure	pressure [dbar]
longitude	longitude of observation (only used if eos="gsw"; see 'Details').
latitude	latitude of observation (only used if eos="gsw"; see 'Details').
eos	equation of state, either "unesco" [1,2] or "gsw" [3,4].

Details

Definition: $\sigma_1 = \sigma_\theta = \rho(S, \theta(S, t, p), 1000$ - 1000 kg/m^3.

Value

Potential density anomaly [kg/m^3].

Author(s)

Dan Kelley

References

See citations provided in the swRho documentation.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

---

swSigma2 *Seawater potential density anomaly referenced to 2000db pressure*

Description

Compute σ_θ, the potential density of seawater (minus 1000 kg/m^3), referenced to 2000db pressure.

Usage

```
swSigma2(salinity, temperature = NULL, pressure = NULL, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

salinity	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an `oce` object, in which case `salinity`, `temperature` (in the ITS-90 scale; see next item), etc. are inferred from the object.
temperature	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting `eos="gsw"`), and is the value contained within `ctd` objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using T68fromT90.
pressure	pressure [dbar]
longitude	longitude of observation (only used if `eos="gsw"`; see 'Details').
latitude	latitude of observation (only used if `eos="gsw"`; see 'Details').
eos	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

Definition: $\sigma_1 = \sigma_\theta = \rho(S, \theta(S, t, p),$ 1000 - 2000 kg/m^3.

Value

Potential density anomaly [kg/m^3].

Author(s)

Dan Kelley

References

See citations provided in the swRho documentation.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

---

swSigma3 *Seawater potential density anomaly referenced to 3000db pressure*

Description

Compute σ_θ, the potential density of seawater (minus 1000 kg/m^3), referenced to 3000db pressure.

Usage

```
swSigma3(salinity, temperature = NULL, pressure = NULL, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

salinity	either practical salinity (in which case temperature and pressure must be provided) **or** an oce object, in which case salinity, temperature (in the ITS-90 scale; see next item), etc. are inferred from the object.
temperature	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting eos="gsw"), and is the value contained within ctd objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using T68fromT90.
pressure	pressure [dbar]
longitude	longitude of observation (only used if eos="gsw"; see 'Details').
latitude	latitude of observation (only used if eos="gsw"; see 'Details').
eos	equation of state, either "unesco" [1,2] or "gsw" [3,4].

Details

Definition: $\sigma_1 = \sigma_\theta = \rho(S, \theta(S, t, p), 3000$ - 1000 kg/m^3.

Value

Potential density anomaly [kg/m^3].

Author(s)

Dan Kelley

References

See citations provided in the swRho documentation.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

---

swSigma4 *Seawater potential density anomaly referenced to 4000db pressure*

Description

Compute σ_θ, the potential density of seawater (minus 1000 kg/m^3), referenced to 4000db pressures.

Usage

```
swSigma4(salinity, temperature = NULL, pressure = NULL, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

salinity	either practical salinity (in which case temperature and pressure must be provided) **or** an oce object, in which case salinity, temperature (in the ITS-90 scale; see next item), etc. are inferred from the object.
temperature	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting eos="gsw"), and is the value contained within ctd objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using T68fromT90.
pressure	pressure [dbar]
longitude	longitude of observation (only used if eos="gsw"; see 'Details').
latitude	latitude of observation (only used if eos="gsw"; see 'Details').
eos	equation of state, either "unesco" [1,2] or "gsw" [3,4].

Details

Definition: $\sigma_1 = \sigma_\theta = \rho(S, \theta(S, t, p), 4000$ - 1000 kg/m^3.

Value

Potential density anomaly [kg/m^3].

Author(s)

Dan Kelley

References

See citations provided in the swRho documentation.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

---

swSigmaT *Seawater quasi-potential density anomaly*

Description

Compute σ_t, a rough estimate of potential density of seawater, minus 1000 kg/m^3.

Usage

```
swSigmaT(salinity, temperature = NULL, pressure = NULL, longitude = NULL,
  latitude = NULL, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an oce object, in which case `salinity`, `temperature` (in the ITS-90 scale; see next item), etc. are inferred from the object.
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting `eos="gsw"`), and is the value contained within `ctd` objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using T68fromT90.
`pressure`	pressure [dbar]

`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

If the first argument is an `oce` object, then salinity, etc., are extracted from it, and used for the calculation.

Value

Quasi-potential density anomaly [kg/m^3], defined as the density calculated with pressure set to zero.

Author(s)

Dan Kelley

References

See citations provided in the `swRho` documentation.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
swSigmaT(35, 13, 1000, longitude=300, latitude=30, eos="gsw") # 26.39623
swSigmaT(35, T90fromT68(13), 1000, eos="unesco") # 26.39354
```

`swSigmaTheta` *Seawater potential density anomaly*

Description

Compute the potential density (minus 1000 kg/m^3) that seawater would have if raised adiabatically to the surface. In the UNESCO system, this quantity is is denoted σ_θ (hence the function name), but in the GSW system, it is denoted `sigma0`.

Usage

```
swSigmaTheta(salinity, temperature = NULL, pressure = NULL,
  referencePressure = 0, longitude = NULL, latitude = NULL,
  eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an `oce` object, in which case `salinity`, `temperature` (in the ITS-90 scale; see next item), etc. are inferred from the object.
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting `eos="gsw"`), and is the value contained within `ctd` objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using `T68fromT90`.
`pressure`	pressure [dbar]
`referencePressure`	The reference pressure, in dbar.
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

If the first argument is an `oce` object, then salinity, etc., are extracted from it, and used for the calculation instead of any values provided in the other arguments.

Value

Potential density anomaly [kg/m^3], defined as $\sigma_\theta = \rho(S, \theta(S, t, p), 0$ - 1000 kg/m^3.

Author(s)

Dan Kelley

References

See citations provided in the `swRho` documentation.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
expect_equal(26.4212790994, swSigmaTheta(35, 13, 1000, eos="unesco"))
```

`swSoundAbsorption` *Seawater sound absorption in dB/m*

Description

Compute the sound absorption of seawater, in dB/m

Usage

```
swSoundAbsorption(frequency, salinity, temperature, pressure, pH = 8,
  formulation = c("fisher-simmons", "francois-garrison"))
```

Arguments

`frequency`	The frequency of sound, in Hz.
`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an `oce` object, in which case `salinity`, `temperature` (in the ITS-90 scale; see next item), etc. are inferred from the object.
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting `eos="gsw"`), and is the value contained within `ctd` objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using `T68fromT90`.
`pressure`	pressure [dbar]
`pH`	seawater pH
`formulation`	character string indicating the formulation to use, either of `"fischer-simmons"` or `"francois-garrison"`; see "References".

Details

Salinity and pH are ignored in this formulation. Several formulae may be found in the literature, and they give results differing by 10 percent, as shown at [3] for example. For this reason, it is likely that more formulations will be added to this function, and entirely possible that the default may change.

Value

Sound absorption in dB/m.

Author(s)

Dan Kelley

References

1. F. H. Fisher and V. P. Simmons, 1977. Sound absorption in sea water. J. Acoust. Soc. Am., 62(3), 558-564.

2. R. E. Francois and G. R. Garrison, 1982. Sound absorption based on ocean measurements. Part II: Boric acid contribution and equation for total absorption. J. Acoust. Soc. Am., 72(6):1879-1890.

3. http://resource.npl.co.uk/acoustics/techguides/seaabsorption/

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
## Fisher & Simmons (1977 table IV) gives 0.52 dB/km for 35 PSU, 5 degC, 500 atm
## (4990 dbar of water)a and 10 kHz
alpha <- swSoundAbsorption(35, 4, 4990, 10e3)

## reproduce part of Fig 8 of Francois and Garrison (1982 Fig 8)
f <- 1e3 * 10^(seq(-1,3,0.1)) # in KHz
plot(f/1000, 1e3*swSoundAbsorption(f, 35, 10, 0, formulation='fr'),
     xlab=" Freq [kHz]", ylab=" dB/km", type='l', log='xy')
lines(f/1000, 1e3*swSoundAbsorption(f, 0, 10, 0, formulation='fr'), lty='dashed')
legend("topleft", lty=c("solid", "dashed"), legend=c("S=35", "S=0"))
```

---

swSoundSpeed *Seawater sound speed*

Description

Compute the seawater speed of sound.

Usage

```
swSoundSpeed(salinity, temperature = NULL, pressure = NULL,
  longitude = NULL, latitude = NULL, eos = getOption("oceEOS", default =
  "gsw"))
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an `oce` object, in which case `salinity`, `temperature` (in the ITS-90 scale; see next item), etc. are inferred from the object.
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale. This scale is used by GSW-style calculation (as requested by setting `eos="gsw"`), and is the value contained within `ctd` objects (and probably most other objects created with data acquired in the past decade or two). Since the UNESCO-style calculation is based on IPTS-68, the temperature is converted within the present function, using `T68fromT90`.
`pressure`	pressure [dbar]
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

If `eos="unesco"`, the sound speed is calculated using the formulation in section 9 of Fofonoff and Millard (1983). If `eos="gsw"`, then the `gsw_sound_speed` function from the `gsw` package is used.

Value

Sound speed [m/s].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp. (See section 9.)

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
swSoundSpeed(40, T90fromT68(40), 10000) # 1731.995 (p48 of Fofonoff + Millard 1983)
```

swSpecificHeat *Seawater specific heat Source= http://sam.ucsd.edu/sio210/propseawater/ppsw_fortran/ppsw.f check value: cpsw = 3849.500 j/(kg deg. c) for s = 40 (ipss-78),*

Description

Compute specific heat of seawater.

Usage

```
swSpecificHeat(salinity, temperature = NULL, pressure = 0,
  longitude = NULL, latitude = NULL, eos = getOption("oceEOS", default =
  "gsw"))
```

Arguments

`salinity`	either practical salinity (in which case `temperature` and `pressure` must be provided) **or** an `oce` object (in which case `salinity`, etc. are inferred from the object).
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale.
`pressure`	seawater pressure [dbar]
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` or `"gsw"`.

Details

If the first argument is a `ctd` object, then salinity, etc, are extracted from it, and used for the calculation.

Value

Specific heat $Jkg^{-1}\,{}^{\circ}C^{-1}$

Author(s)

Dan Kelley

References

Millero et. al., J. Geophys. Res. 78 (1973), 4499-4507

Millero et. al., UNESCO report 38 (1981), 99-188.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
swSpecificHeat(40, T90fromT68(40), 10000, eos="unesco") # 3949.499
```

---

swSpice *Seawater spiciness*

Description

Compute seawater "spice" (a variable orthogonal to density in TS space).

Usage

```
swSpice(salinity, temperature = NULL, pressure = NULL)
```

Arguments

salinity	either salinity [PSU] (in which case temperature and pressure must be provided) **or** a ctd object (in which case salinity, temperature and pressure are determined from the object, and must not be provided in the argument list).
temperature	*in-situ* temperature [°C] on the ITS-90 scale; see "Temperature units" in the documentation for swRho.
pressure	seawater pressure [dbar]

Details

If the first argument is a ctd object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Roughly speaking, seawater with a high spiciness is relatively warm and salty compared with less spicy water. Another interpretation is that spice is a variable measuring distance orthogonal to isopycnal lines on TS diagrams (if the diagrams are scaled to make the isopycnals run at 45 degrees). The definition used here is that of Pierre Flament. (Other formulations exist.) Note that pressure is ignored in the definition. Spiciness is sometimes denoted $\pi(S, t, p)$.

Value

Spice [kg/m^3].

Author(s)

Dan Kelley

References

P. Flament, 2002. A state variable for characterizing water masses and their diffusive stability: spiciness. *Progr. Oceanog.*, **54**, 493-501.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

---

swSTrho *Seawater salinity from temperature and density*

Description

Compute Practical or Absolute Salinity, given in-situ or Conservative Temperature, density, and pressure. This is mainly used to draw isopycnal lines on TS diagrams, hence the dual meanings for salinity and temperature, depending on the value of `eos`.

Usage

```
swSTrho(temperature, density, pressure, eos = getOption("oceEOS", default =
  "gsw"))
```

Arguments

`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale; see “Temperature units” in the documentation for `swRho`.
`density`	*in-situ* density or sigma value [kg/m^3]
`pressure`	*in-situ* pressure [dbar]
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

For `eos="unesco"`, finds the practical salinity that yields the given density, with the given in-situ temperature and pressure. The method is a bisection search with a salinity tolerance of 0.001. For `eos="gsw"`, the function `gsw_SA_from_rho` in the gsw package is used to infer Absolute Salinity from Conservative Temperature.

Value

Practical Salinity, if eos="unesco", or Absolute Salinity, if eos="gsw".

Author(s)

Dan Kelley

References

1. Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

2. Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

3. IOC, SCOR, and IAPSO (2010). The international thermodynamic equation of seawater-2010: Calculation and use of thermodynamic properties. Technical Report 56, Intergovernmental Oceanographic Commission, Manuals and Guide.

4. McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

See Also

swTSrho

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
swSTrho(10, 22, 0, eos="gsw") # 28.76285
swSTrho(10, 22, 0, eos="unesco") # 28.651625
```

---

swTFreeze *Seawater freezing temperature*

Description

Compute freezing temperature of seawater.

Usage

```
swTFreeze(salinity, pressure = 0, longitude = NULL, latitude = NULL,
  saturation_fraction = 1, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity`	either salinity [PSU] or a `ctd` object from which salinity will be inferred.
`pressure`	seawater pressure [dbar]
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`saturation_fraction`	saturation fraction of dissolved air in seawater (used only if `eos="gsw"`).
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

In the first form, the argument is a `ctd` object, from which the salinity and pressure values are extracted and used to for the calculation.

Value

Temperature [°C], defined on the ITS-90 scale.

Author(s)

Dan Kelley

References

[1] Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

[2] Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

[3] IOC, SCOR, and IAPSO (2010). The international thermodynamic equation of seawater-2010: Calculation and use of thermodynamic properties. Technical Report 56, Intergovernmental Oceanographic Commission, Manuals and Guide.

[4] McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
swTFreeze(salinity=40, pressure=500, eos="unesco") # -2.588567 degC
```

swThermalConductivity *Seawater thermal conductivity*

Description

Compute seawater thermal conductivity, in $Wm^{-1\circ}C^{-1}$

Usage

```
swThermalConductivity(salinity, temperature = NULL, pressure = NULL)
```

Arguments

`salinity`	salinity [PSU], or a `ctd` object, in which case `temperature` and `pressure` will be ignored.
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale; see "Temperature units" in the documentation for `swRho`.
`pressure`	pressure [dbar]

Details

Caldwell's (1974) detailed formulation is used. To be specific, his equation 6 to calculate K, and his two sentences above that equation are used to infer this to be K(0,T,S) in his notation of equation 7. Then, application of his equations 7 and 8 is straightforward. He states an accuracy for this method of 0.3 percent. (See the check against his Table 1 in the "Examples".)

Value

Conductivity of seawater in $Wm^{-1\circ}C^{-1}$.

To calculate thermal diffusivity in m^2/s, divide by the product of density and specific heat, as in the example.

Author(s)

Dan Kelley

References

Caldwell, Douglas R., 1974. Thermal conductivity of seawater, *Deep-sea Research*, **21**, 131-137.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swTheta`, `swViscosity`, `swZ`

Examples

```
library(oce)
# Values in m^2/s, a unit that is often used instead of W/(m*degC).
swThermalConductivity(35, 10, 100) / (swRho(35,10,100) * swSpecificHeat(35,10,100)) # ocean
swThermalConductivity(0, 20, 0) / (swRho(0, 20, 0) * swSpecificHeat(0, 20, 0)) # lab
# Caldwell Table 1 gives 1478e-6 cal/(cm*sec*degC) at 31.5 o/oo, 10degC, 1kbar
joulePerCalorie <- 4.18400
cmPerM <- 100
swThermalConductivity(31.5,10,1000) / joulePerCalorie / cmPerM
```

---

swTheta *Seawater potential temperature*

Description

Compute the potential temperature of seawater, denoted θ in the UNESCO system, and `pt` in the GSW system.

Usage

```
swTheta(salinity, temperature = NULL, pressure = NULL,
  referencePressure = 0, longitude = NULL, latitude = NULL,
  eos = getOption("oceEOS", default = "gsw"))
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** an `oce` object (in which case `salinity`, etc. are inferred from the object).
`temperature`	*in-situ* temperature [°C], defined on the ITS-90 scale; see "Temperature units" in the documentation for `swRho`, and the examples below.
`pressure`	pressure [dbar]
`referencePressure`	reference pressure [dbar]
`longitude`	longitude of observation (only used if `eos="gsw"`; see 'Details').
`latitude`	latitude of observation (only used if `eos="gsw"`; see 'Details').
`eos`	equation of state, either `"unesco"` [1,2] or `"gsw"` [3,4].

Details

Different formulae are used depending on the equation of state. If `eos` is `"unesco"`, the method of Fofonoff *et al.* (1983) is used [1,2]. Otherwise, `swTheta` uses `gsw_pt_from_t` from the `gsw` package.

If the first argument is a `ctd` or `section` object, then values for salinity, etc., are extracted from it, and used for the calculation, and the corresponding arguments to the present function are ignored.

Value

Potential temperature [°C] of seawater, referenced to pressure `referencePressure`.

Author(s)

Dan Kelley

References

[1] Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

[2] Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

[3] IOC, SCOR, and IAPSO (2010). The international thermodynamic equation of seawater-2010: Calculation and use of thermodynamic properties. Technical Report 56, Intergovernmental Oceanographic Commission, Manuals and Guide.

[4] McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swViscosity`, `swZ`

Examples

```
library(oce)
## test value from Fofonoff et al., 1983
expect_equal(36.8818748026, swTheta(40, T90fromT68(40), 10000, 0, eos="unesco"))

# Example from a cross-Atlantic section
data(section)
stn <- section[['station', 70]]
plotProfile(stn, 'theta', ylim=c(6000, 1000))
lines(stn[['temperature']], stn[['pressure']], lty=2)
legend("bottomright", lty=1:2,
       legend=c("potential", "in-situ"),
       bg='white', title="Station 70")
```

swTSrho *Seawater temperature from salinity and density*

Description

Compute *in-situ* temperature, given salinity, density, and pressure.

Usage

```
swTSrho(salinity, density, pressure = NULL, eos = getOption("oceEOS",
  default = "gsw"))
```

Arguments

salinity	*in-situ* salinity [PSU]
density	*in-situ* density or sigma value [kg/m^3]
pressure	*in-situ* pressure [dbar]
eos	equation of state to be used, either "unesco" or "gsw" (ignored at present).

Details

Finds the temperature that yields the given density, with the given salinity and pressure. The method is a bisection search with temperature tolerance 0.001 $^\circ C$.

Value

In-situ temperature [$^\circ C$] in the ITS-90 scale.

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

See Also

swSTrho

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swThermalConductivity, swTheta, swViscosity, swZ

Examples

```
swTSrho(35, 23, 0, eos="unesco") # 26.11301
```

---

swViscosity *Seawater viscosity*

Description

Compute viscosity of seawater, in $Pa \cdot s$

Usage

```
swViscosity(salinity, temperature)
```

Arguments

salinity	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
temperature	*in-situ* temperature [°C], defined on the ITS-90 scale; see "Temperature units" in the documentation for `swRho`, and the examples below.

Details

If the first argument is a `ctd` object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

The result is determined from a regression of the data provided in Table 87 of Dorsey (1940). The fit matches the table to within 0.2 percent at worst, and with average absolute error of 0.07 percent. The maximum deviation from the table is one unit in the last decimal place.

No pressure dependence was reported by Dorsey (1940).

Value

Viscosity of seawater in $Pa \cdot s$. Divide by density to get kinematic viscosity in m^2/s.

Author(s)

Dan Kelley

References

N. Ernest Dorsey (1940), *Properties of ordinary Water-substance*, American Chemical Society Monograph Series. Reinhold Publishing.

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swZ

Examples

```
swViscosity(30, 10) # 0.001383779
```

---

swZ *Vertical coordinate*

Description

Compute height above the surface. This is the negative of depth, and so is defined simply in terms of swDepth.

Usage

```
swZ(pressure, latitude = 45, eos = getOption("oceEOS", default = "gsw"))
```

Arguments

pressure	either pressure [dbar], in which case `lat` must also be given, or a `ctd` object, in which case `lat` will be inferred from the object.
latitude	Latitude in °N or radians north of the equator.
eos	indication of formulation to be used, either "unesco" or "gsw".

See Also

Other functions that calculate seawater properties: T68fromT90, T90fromT48, T90fromT68, swAbsoluteSalinity, swAlphaOverBeta, swAlpha, swBeta, swCSTp, swConservativeTemperature, swDepth, swDynamicHeight, swLapseRate, swN2, swPressure, swRho, swRrho, swSCTp, swSTrho, swSigma0, swSigma1, swSigma2, swSigma3, swSigma4, swSigmaTheta, swSigmaT, swSigma, swSoundAbsorption, swSoundSpeed, swSpecificHeat, swSpice, swTFreeze, swTSrho, swThermalConductivity, swTheta, swViscosity

T68fromT90 *Convert from ITS-90 to IPTS-68 temperature*

Description

Today's instruments typically record in the ITS-90 scale, but some old datasets will be in the IPTS-68 scale. `T90fromT68()` converts from the IPTS-68 to the ITS-90 scale, using Saunders' (1990) formula, while `T68fromT90()` does the reverse. The difference between IPTS-68 and ITS-90 values is typically a few millidegrees (see 'Examples'), which is seldom visible on a typical temperature profile, but may be of interest in some precise work. Mostly for historical interest, `T90fromT48()` is provided to convert from the ITS-48 system to ITS-90.

Usage

```
T68fromT90(temperature)
```

Arguments

`temperature`	Vector of temperatures expressed in the ITS-90 scale.

Value

Temperature expressed in the IPTS-68 scale.

Author(s)

Dan Kelley

References

P. M. Saunders, 1990. The international temperature scale of 1990, ITS-90. WOCE Newsletter, volume 10, September 1990, page 10. (`http://www.nodc.noaa.gov/woce/wdiu/wocedocs/newsltr/news10/contents.htm`)

See Also

Other functions that calculate seawater properties: `T90fromT48`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
library(oce)
T68 <- seq(3, 20, 1)
T90 <- T90fromT68(T68)
sqrt(mean((T68-T90)^2))
```

T90fromT48 *Convert from ITS-48 to ITS-90 temperature*

Description

Today's instruments typically record in the ITS-90 scale, but some old datasets will be in the IPTS-68 scale. `T90fromT68()` converts from the IPTS-68 to the ITS-90 scale, using Saunders' (1990) formula, while `T68fromT90()` does the reverse. The difference between IPTS-68 and ITS-90 values is typically a few millidegrees (see 'Examples'), which is seldom visible on a typical temperature profile, but may be of interest in some precise work. Mostly for historical interest, `T90fromT48()` is provided to convert from the ITS-48 system to ITS-90.

Usage

```
T90fromT48(temperature)
```

Arguments

`temperature`	Vector of temperatures expressed in the ITS-48 scale.

Value

Temperature expressed in the ITS-90 scale.

Author(s)

Dan Kelley

References

P. M. Saunders, 1990. The international temperature scale of 1990, ITS-90. WOCE Newsletter, volume 10, September 1990, page 10. (http://www.nodc.noaa.gov/woce/wdiu/wocedocs/newsltr/news10/contents.htm)

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT68`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
library(oce)
T68 <- seq(3, 20, 1)
T90 <- T90fromT68(T68)
sqrt(mean((T68-T90)^2))
```

T90fromT68 *Convert from IPTS-68 to ITS-90 temperature*

Description

Today's instruments typically record in the ITS-90 scale, but some old datasets will be in the IPTS-68 scale. `T90fromT68()` converts from the IPTS-68 to the ITS-90 scale, using Saunders' (1990) formula, while `T68fromT90()` does the reverse. The difference between IPTS-68 and ITS-90 values is typically a few millidegrees (see 'Examples'), which is seldom visible on a typical temperature profile, but may be of interest in some precise work. Mostly for historical interest, `T90fromT48()` is provided to convert from the ITS-48 system to ITS-90.

Usage

```
T90fromT68(temperature)
```

Arguments

`temperature`	Vector of temperatures expressed in the IPTS-68 scale.

Value

temperature Temperature expressed in the ITS-90 scale.

Author(s)

Dan Kelley

References

P. M. Saunders, 1990. The international temperature scale of 1990, ITS-90. WOCE Newsletter, volume 10, September 1990, page 10. (http://www.nodc.noaa.gov/woce/wdiu/wocedocs/newsltr/news10/contents.htm)

See Also

Other functions that calculate seawater properties: `T68fromT90`, `T90fromT48`, `swAbsoluteSalinity`, `swAlphaOverBeta`, `swAlpha`, `swBeta`, `swCSTp`, `swConservativeTemperature`, `swDepth`, `swDynamicHeight`, `swLapseRate`, `swN2`, `swPressure`, `swRho`, `swRrho`, `swSCTp`, `swSTrho`, `swSigma0`, `swSigma1`, `swSigma2`, `swSigma3`, `swSigma4`, `swSigmaTheta`, `swSigmaT`, `swSigma`, `swSoundAbsorption`, `swSoundSpeed`, `swSpecificHeat`, `swSpice`, `swTFreeze`, `swTSrho`, `swThermalConductivity`, `swTheta`, `swViscosity`, `swZ`

Examples

```
library(oce)
T68 <- seq(3, 20, 1)
T90 <- T90fromT68(T68)
sqrt(mean((T68-T90)^2))
```

---

tail.oce   *Extract the End of an Oce Object*

Description

Extract the End of an Oce Object

Usage

```
## S3 method for class 'oce'
tail(x, n = 6L, ...)
```

Arguments

x	An oce object of a suitable class (presently only adp is permitted).
n	Number of elements to extract.
...	ignored

See Also

head.oce, which yields the start of an oce object.

---

threenum   *Calculate min, mean, and max values*

Description

This is a faster cousin of the standard fivenum function, used in generic summary functions for oce objects.

Usage

```
threenum(x)
```

Arguments

x	a vector or matrix of numerical values.

Value

A character vector of four values: the minimum, the mean, the maximum, and an indication of the number of data.

Author(s)

Dan Kelley

Examples

```
library(oce)
threenum(1:10)
```

---

tidedata *Tidal Constituent Information*

Description

The `tidedata` dataset contains Tide-constituent information that is use by `tidem` to fit tidal models. `tidedata` is a list containing

`const` a list containing vectors `name` (a string with constituent name), `freq` (the frequency, in cycles per hour), `kmpr` (a string naming the comparison constituent, blank if there is none), `ikmpr` (index of comparison constituent, or `0` if there is none), `df` (frequency difference between constituent and its comparison, used in the Rayleigh criterion), `d1` through `d6` (the first through sixth Doodson numbers), `semi`, `nsat` (number of satellite constituents), `ishallow`, `nshallow`, `doodsonamp`, and `doodsonspecies`.

`sat` a list containing vectors `deldood`, `phcorr`, `amprat`, `ilatfac`, and `iconst`.

`shallow` a list containing vectors `iconst`, `coef`, and `iname`.

Apart from the use of `d1` through `d6`, the naming and content follows `T_TIDE` (see Pawlowicz et al. 2002), which in turn builds upon the analysis of Foreman (1977).

Author(s)

Dan Kelley

Source

The data come from the `tide3.dat` file of the `T_TIDE` package (Pawlowicz et al., 2002), and derive from Appendices provided by Foreman (1977). The data are scanned using '`tests/tide.R`' in this package, which also performs some tests using `T_TIDE` values as a reference.

References

Foreman, M. G. G., 1977. Manual for tidal heights analysis and prediction. Pacific Marine Science Report 77-10, Institute of Ocean Sciences, Patricia Bay, Sidney, BC, 58pp.

Pawlowicz, Rich, Bob Beardsley, and Steve Lentz, 2002. Classical tidal harmonic analysis including error estimates in MATLAB using `T_TIDE`. Computers and Geosciences, 28, 929-937.

See Also

Other things related to `tidem` data: `[[,tidem-method`, `[[<-,tidem-method`, `plot,tidem-method`, `predict.tidem`, `summary,tidem-method`, `tidem-class`, `tidemAstron`, `tidemVuf`, `tidem`

---

tidem — *Fit a Tidem (Tidal Model) to a Timeseries*

Description

The fit is done in terms of sine and cosine components at the indicated tidal frequencies, with the amplitude and phase being calculated from the resultant coefficients on the sine and cosine terms.

Usage

```
tidem(t, x, constituents, infer = NULL, latitude = NULL, rc = 1,
  regress = lm, debug = getOption("oceDebug"))
```

Arguments

`t`	Either a `sealevel` object (e.g. produced by `read.sealevel` or `as.sealevel`) or a vector of times. In the former case, time is part of the object, so `t` may not be given here. In the latter case, `tidem` needs a way to determine time, so `t` must be given.
`x`	an optional numerical vector holding something that varies with time. This is ignored if `t` is a `sealevel-class` object, in which case it is inferred as `t[["elevation"]]`.
`constituents`	an optional vector of strings that name tidal constituents to which the fit is done (see "Details" and "Constituent Naming Convention".)
`infer`	a list of constituents to be inferred from fitted constituents according to the method outlined in Section 2.3.4 of Foreman (1977) [1]. If `infer` is `NULL`, the default, then no such inferences are made. Otherwise, some constituents are computed based on other constituents, instead of being determined by regression at the proper frequency. If provided, `infer` must be a list containing four elements: `name`, a vector of strings naming the constituents to be inferred; `from`, a vector of strings naming the fitted constituents used as the sources for those inferences (these source constituents are added to the regression list, if they are not already there); `amp`, a numerical vector of factors to be applied to the source amplitudes; and `phase`, a numerical vector of angles, in degrees, to be subtracted from the source phases. For example, Following Foreman (1997) [1], if any of the `name` items have already been computed, then the suggested inference is ignored, and the already-computed values are used.

```
infer=list(name=c("P1","K2"),
           from=c("K1", "S2"),
           amp=c(0.33093, 0.27215),
           phase=c(-7.07, -22.4)
```

means that the amplitude of `P1` will be set as 0.33093 times the calculated amplitude of `K1`, and that the `P1` phase will be set to the `K1` phase, minus an offset of `-7.07` degrees. (This example is used in the Foreman (1977) [1] discussion of a Fortran analysis code and also in Pawlowicz et al. (2002) [4] discussion of the T_TIDE Matlab code. Rounded to the 0.1mm resolution of values reported in [1] and [2], the `tidem` results have root-mean-square amplitude difference to Foreman's Appendix 7.3 of 0.06mm; by comparison, the results in Table 1 of Pawlowicz et al. (2002) agree with Foreman's results to RMS difference 0.04mm.)

`latitude` if provided, the latitude of the observations. If not provided, `tidem` will try to infer this from `sl`.

`rc` the value of the coefficient in the Rayleigh criterion.

`regress` function to be used for regression, by default `lm`, but could be for example `rlm` from the `MASS` package.

`debug` an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

The tidal constituents to be used in the analysis are specified as follows; see "Constituent Naming Convention".

- **Case 1**. If `constituents` is not provided, then the constituent list will be made up of the 69 constituents designated by Foreman as "standard". These include astronomical frequencies and some shallow-water frequencies, and are as follows: `c("Z0", "SA", "SSA", "MSM", "MM", "MSF", "MF","ALP1`
- **Case 2**. If the first item in `constituents` is the string `"standard"`, then a provisional list is set up as in Case 1, and then the (optional) rest of the elements of `constituents` are examined, in order. Each of these constituents is based on the name of a tidal constituent in the Foreman (1977) notation. (To get the list, execute `data(tidedata)` and then execute `cat(tideData$name)`.) Each named constituent is added to the existing list, if it is not already there. But, if the constituent is preceded by a minus sign, then it is removed from the list (if it is already there). Thus, for example, `constituents=c("standard", "-M2", "ST32")` would remove the M2 constituent and add the ST32 constituent.
- **Case 3**. If the first item is not `"standard"`, then the list of constituents is processed as in Case 2, but without starting with the standard list. As an example, `constituents=c("K1", "M2")` would fit for just the K1 and M2 components. (It would be strange to use a minus sign to remove items from the list, but the function allows that.)

In each of the above cases, the list is reordered in frequency prior to the analysis, so that the results of `summary,tidem-method` will be in a familiar form.

Once the constituent list is determined, `tidem` prunes the elements of the list by using the Rayleigh criterion, according to which two constituents of frequencies f_1 and f_2 cannot be resolved unless the time series spans a time interval of at least $rc/(f_1 - f_2)$.

Finally, `tidem` looks in the remaining constituent list to check that the application of the Rayleigh criterion has not removed any of the constituents specified directly in the `constituents` argument. If any are found to have been removed, then they are added back. This last step was added on 2017-12-27, to make `tidem` behave the same way as the Foreman (1977) code [1], as illustrated in his Appendices 7.2 and 7.3. (As an aside, his Appendix 7.3 has some errors, e.g. the frequency for the 2SK5 constituent is listed there (p58) as 0.20844743, but it is listed as 0.2084474129 in his Appendix 7.1 (p41). For this reason, the frequency comparison is relaxed to a `tol` value of `1e-7` in a portion of the oce test suite (see `tests/testthat/test_tidem.R` in the source).

A specific example may be of help in understanding the removal of unresolvable constitutents. For example, the `data(sealevel)` dataset is of length 6718 hours, and this is too short to resolve the full list of constituents, with the conventional (and, really, necessary) limit of `rc=1`. From Table 1 of [1], this timeseries is too short to resolve the `SA` constituent, so that `SA` will not be in the resultant. Similarly, Table 2 of [1] dictates the removal of `PI1`, `S1` and `PSI1` from the list. And, finally, Table 3 of [1] dictates the removal of `H1`, `H2`, `T2` and `R2`. Also, since Table 3 of [1] indicates that `GAM2` gets subsumed into `H1`, and if `H1` is already being deleted in this test case, then `GAM2` will also be deleted.

A list of constituent names is created by the following:

```
data(tidedata)
print(tidedata$const$name)
```

The text should include discussion of the (not yet performed) nodal correction treatment.

Value

An object of `tidem-class`, consisting of

`const`	constituent number, e.g. 1 for `Z0`, 1 for `SA`, etc.
`model`	the regression model
`name`	a vector of constituent names, in non-subscript format, e.g. `"M2"`.
`frequency`	a vector of constituent frequencies, in inverse hours.
`amplitude`	a vector of fitted constituent amplitudes, in metres.
`phase`	a vector of fitted constituent phase. NOTE: The definition of phase is likely to change as this function evolves. For now, it is phase with respect to the first data sample.
`p`	a vector containing a sort of `p` value for each constituent. This is calculated as the average of the `p` values for the sine() and cosine() portions used in fitting; whether it makes any sense is an open question.

Bugs

1. This function is not fully developed yet, and both the form of the call and the results of the calculation may change.
2. Nodal correction is not done.
3. The reported `p` value may make no sense at all, and it might be removed in a future version of this function. Perhaps a significance level should be presented, as in the software developed by both Foreman and Pawlowicz.

Constituent Naming Convention

tidem uses constituent names that follow the convention set by Foreman (1977) [1]. This convention is slightly different from that used in the T-TIDE package of Pawlowicz et al. (2002) [4], with Foreman's UPS1 and M8 becoming UPSI and MS in T-TIDE. As a convenience, tidem converts from these T-TIDE names to the Foreman names, issuing warnings when doing so.

Agreement with T_TIDE results

The tidem amplitude and phase results, obtained with

```
tidem(sealevelTuktoyaktuk, constituents=c("standard", "M10"),
     infer=list(name=c("P1", "K2"),
                from=c("K1", "S2"),
                amp=c(0.33093, 0.27215),
                phase=c(-7.07, -22.40))),
```

are identical the T_TIDE values listed in Table 1 of Pawlowicz et al. (2002), after rounding amplitude and phase to 4 and 2 digits past the decimal place, to match the format of the table.

Author(s)

Dan Kelley

References

1. Foreman, M. G. G., 1977. Manual for tidal heights analysis and prediction. Pacific Marine Science Report 77-10, Institute of Ocean Sciences, Patricia Bay, Sidney, BC, 58pp.

2. Foreman, M. G. G., Neufeld, E. T., 1991. Harmonic tidal analyses of long time series. International Hydrographic Review, 68 (1), 95-108.

3. Leffler, K. E. and D. A. Jay, 2009. Enhancing tidal harmonic analysis: Robust (hybrid) solutions. Continental Shelf Research, 29(1):78-88.

4. Pawlowicz, Rich, Bob Beardsley, and Steve Lentz, 2002. Classical tidal harmonic analysis including error estimates in MATLAB using T_TIDE. Computers and Geosciences, 28, 929-937.

See Also

Other things related to tidem data: [[,tidem-method, [[<-,tidem-method, plot,tidem-method, predict.tidem, summary,tidem-method, tidedata, tidem-class, tidemAstron, tidemVuf

Examples

```
library(oce)
# The demonstration time series from Foreman (1977),
# also used in T_TIDE (Pawlowicz, 2002).
data(sealevelTuktoyaktuk)
tide <- tidem(sealevelTuktoyaktuk)
summary(tide)

# AIC analysis
```

```
extractAIC(tide[["model"]])

# Fake data at M2
t <- seq(0, 10*86400, 3600)
eta <- sin(0.080511401 * t * 2 * pi / 3600)
sl <- as.sealevel(eta)
m <- tidem(sl)
summary(m)
```

`tidem-class` *Class to Store Tidal Models*

Description

Class to store tidal-constituent models.

Author(s)

Dan Kelley

See Also

Other functions that plot oce data: plot,adp-method, plot,adv-method, plot,amsr-method, plot,argo-method, plot,bremen-method, plot,cm-method, plot,coastline-method, plot,ctd-method, plot,gps-method, plot,ladp-method, plot,lisst-method, plot,lobo-method, plot,met-method, plot,odf-method, plot,rsk-method, plot,satellite-method, plot,sealevel-method, plot,section-method, plot,tidem-method, plot,topo-method, plot,windrose-method, plotProfile, plotScan, plotTS

Other things related to tidem data: [[,tidem-method, [[<-,tidem-method, plot,tidem-method, predict.tidem, summary,tidem-method, tidedata, tidemAstron, tidemVuf, tidem

---

`tidemAstron` *Astronomical Calculations for Tidem*

Description

Do some astronomical calculations for tidem. This function is based directly on t_astron in the T_TIDE Matlab package [1].

Usage

```
tidemAstron(t)
```

Arguments

t	Either a time in POSIXct format (with "UTC" timezone), or an integer. In the second case, it is converted to a time with numberAsPOSIXct(t,tz="UTC"). If t (It is **very** important to use tz="GMT" in constructing t.)

Value

A list containing items named astro and ader (see T_TIDE documentation).

Author(s)

Dan Kelley translated this from t_astron in the T_TIDE package.

References

1. Pawlowicz, Rich, Bob Beardsley, and Steve Lentz, 2002. Classical tidal harmonic analysis including error estimates in MATLAB using T_TIDE. Computers and Geosciences, 28, 929-937.

See Also

Other things related to tidem data: [[,tidem-method, [[<-,tidem-method, plot,tidem-method, predict.tidem, summary,tidem-method, tidedata, tidem-class, tidemVuf, tidem

Examples

```
tidemAstron(as.POSIXct("2008-01-22 18:50:24"))
```

---

tidemVuf — *Nodal Modulation Calculations for Tidem*

Description

Do nodal modulation calculations for tidem. This function is based directly on t_vuf in the T_TIDE Matlab package [1].

Usage

```
tidemVuf(t, j, latitude = NULL)
```

Arguments

t	The time in POSIXct format. (It is **very** important to use tz="GMT" in constructing t.)
j	Indices of tidal constituents to use.
latitude	Optional numerical value containing the latitude in degrees North.

Value

A list containing items named v, u and f (see the T_TIDE documentation).

Author(s)

Dan Kelley translated this from t_astron from the T_TIDE package.

References

1. Pawlowicz, Rich, Bob Beardsley, and Steve Lentz, 2002. Classical tidal harmonic analysis including error estimates in MATLAB using `T_TIDE`. Computers and Geosciences, 28, 929-937.

See Also

Other things related to `tidem` data: `[[,tidem-method`, `[[<-,tidem-method`, `plot,tidem-method`, `predict.tidem`, `summary,tidem-method`, `tidedata`, `tidem-class`, `tidemAstron`, `tidem`

Examples

```
tidemVuf(as.POSIXct("2008-01-22 18:50:24"), 43, 45.0)
```

`titleCase` *Capitalize first letter of each of a vector of words*

Description

This is used in making labels for data names in some ctd functions

Usage

```
titleCase(w)
```

Arguments

`w`	vector of character strings

Value

vector of strings patterned on `w` but with first letter in upper case and others in lower case

---

`toEnu` *Rotate acoustic-Doppler data to the enu coordinate system*

Description

Rotate acoustic-Doppler data to the enu coordinate system

Usage

```
toEnu(x, ...)
```

Arguments

`x`	an adp or adv object, i.e. one inheriting from `adp-class` or `adv-class`.
`...`	extra arguments that are passed on to `toEnuAdp` or `toEnuAdv`.

Value

An object of the same type as `x`, but with velocities in the enu coordinate system

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `velocityStatistics`, `xyzToEnuAdp`, `xyzToEnu`

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOtherAdv`, `enuToOther`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `velocityStatistics`, `xyzToEnuAdv`, `xyzToEnu`

---

toEnuAdp — *Convert an ADP Object to ENU Coordinates*

Description

Convert an ADP Object to ENU Coordinates

Usage

```
toEnuAdp(x, declination = 0, debug = getOption("oceDebug"))
```

Arguments

`x`	an adp object, i.e. one inheriting from `adp-class`.
`declination`	magnetic declination to be added to the heading, to get ENU with N as "true" north.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Author(s)

Dan Kelley

References

http://www.nortek-as.com/lib/forum-attachments/coordinate-transformation

See Also

See read.adp for notes on functions relating to "adp" objects. Also, see beamToXyzAdp and xyzToEnuAdp.

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

---

toEnuAdv — *Convert an ADV Object to ENU Coordinates*

Description

Convert an ADV Object to ENU Coordinates

Usage

```
toEnuAdv(x, declination = 0, debug = getOption("oceDebug"))
```

Arguments

x	An adv object, i.e. one inheriting from adv-class.
declination	magnetic declination to be added to the heading, to get ENU with N as "true" north.
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Author(s)

Dan Kelley

References

http://www.nortek-as.com/lib/forum-attachments/coordinate-transformation

See Also

See read.adv for notes on functions relating to "adv" objects. Also, see beamToXyzAdv and xyzToEnuAdv.

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv-class, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

---

topo-class *Class to Store Topographic Data*

Description

Topographic data may be read with read.topo or assembled with as.topo. Plotting are handled with plot,topo-method and summaries with summary,topo-method. Data retrieval may be done with [[,topo-method and replacement with [[<-,topo-method. The key data, stored within the data slot, are: longititude, latitude, and z.

Author(s)

Dan Kelley

See Also

Other classes provided by oce: adp-class, adv-class, argo-class, bremen-class, cm-class, coastline-class, ctd-class, echosounder-class, lisst-class, lobo-class, met-class, oce-class, odf-class, rsk-class, sealevel-class, section-class, windrose-class

Other things related to topo data: [[,topo-method, [[<-,topo-method, as.topo, download.topo, plot,topo-method, read.topo, subset,topo-method, summary,topo-method, topoInterpolate, topoWorld

---

topoInterpolate *Interpolate Within a Topo Object*

Description

Bilinear interpolation is used so that values will vary smoothly within a longitude-latitude grid cell. Note that the sign convention for longitude and latitude must match that in topo.

Usage

```
topoInterpolate(longitude, latitude, topo)
```

Arguments

`longitude`	Vector of longitudes (in the same sign convention as used in `topo`).
`latitude`	Vector of latitudes (in the same sign convention as used in `topo`).
`topo`	A `topo` object, i.e. inheriting from `topo-class`.

Value

Vector of heights giving the elevation of the earth above means sea level at the indicated location on the earth.

Author(s)

Dan Kelley

See Also

Other things related to `topo` data: `[[,topo-method`, `[[<-,topo-method`, `as.topo`, `download.topo`, `plot,topo-method`, `read.topo`, `subset,topo-method`, `summary,topo-method`, `topo-class`, `topoWorld`

Examples

```
library(oce)
data(topoWorld)
# "The Gully", approx. 400m deep, connects Gulf of St Lawrence with North Atlantic
topoInterpolate(45, -57, topoWorld)
```

`topoWorld` *Global Topographic Dataset at Half-degree Resolution*

Description

Global topographic dataset at half-degree resolution, created by decimating the ETOPO5 dataset. Its longitude ranges from -179.5 to 180, and its latitude ranges from -89.5 to 90. Height is measured in metres above nominal sea level.

The coarse resolution can be a problem in plotting depth contours along with coastlines in regions of steep topography. For example, near the southeast corner of Newfoundland, a 200m contour will overlap a coastline drawn with `coastlineWorldFine`. The solution in such cases is to download a higher-resolution topography file and create a `topo` object with `read.topo` or `as.topo`.

Usage

```
data(topoWorld)
```

Source

The binary ETOPO5 dataset was downloaded in late 2009 from the NOAA website, 'http://www.ngdc.noaa.gov/mgg/glob decoded, decimated from 1/12th degree resolution to 1/2 degree resolution, and passed through matrixShiftLongitude so that longitude is between -180 and 180 degrees.

See Also

Other datasets provided with oce: adp, adv, argo, cm, coastlineWorld, colors, ctdRaw, ctd, echosounder, landsat, lisst, lobo, met, rsk, sealevelTuktoyaktuk, sealevel, section, wind

Other things related to topo data: [[,topo-method, [[<-,topo-method, as.topo, download.topo, plot,topo-method, read.topo, subset,topo-method, summary,topo-method, topo-class, topoInterpolate

Examples

```
## Not run:
library(oce)
data(topoWorld)
par(mfrow=c(2, 1))
plot(topoWorld, location=NULL)
imagep(topoWorld)

## End(Not run)
```

---

trimString *Remove leading and trailing whitespace from strings*

Description

Remove leading and trailing whitespace from strings

Usage

```
trimString(s)
```

Arguments

s	vector of character strings

Value

a new vector formed by trimming leading and trailing whitespace from the elements of s.

unabbreviateYear *Determine year from various abbreviations*

Description

Various data files may contain various abbreviations for years. For example, 99 refers to 1999, and 8 refers to 2008. Sometimes, even 108 refers to 2008 (the idea being that the "zero" year was 1900). This function deals with the three cases mentioned. It will fail if someone supplies 60, meaning year 2060 as opposed to 1960.

Usage

```
unabbreviateYear(year)
```

Arguments

`year`	a year, or vector of years, possibly abbreviated

Author(s)

Dan Kelley

See Also

Other things related to time: `ctimeToSeconds`, `julianCenturyAnomaly`, `julianDay`, `numberAsHMS`, `numberAsPOSIXct`, `secondsToCtime`

Examples

```
fullYear <- unabbreviateYear(c(99, 8, 108))
```

undriftTime *Correct for drift in instrument clock*

Description

It is assumed that the instrument clock matches the real time at the start of the sampling, and that the clock drifts linearly (i.e. is uniformly fast or slow) over the sampling interval. Linear interpolation is used to infer the values of all variables in the `data` slot. The data length is altered in this process, e.g. a slow instrument clock (positive `slowEnd`) takes too few samples in a given time interval, so `undriftTime` will increase the number of data.

Usage

```
undriftTime(x, slowEnd = 0, tname = "time")
```

Arguments

x	an object of oce-class.
slowEnd	number of seconds to add to final instrument time, to get the correct time of the final sample. This will be a positive number, for a "slow" instrument clock.
tname	Character string indicating the name of the time column in the data slot of x.

Value

An object of the same class as x, with the data slot adjusted appropriately.

Author(s)

Dan Kelley

Examples

```
## Not run:
library(oce)
rbr011855 <- read.oce(
 "/data/archive/sleiwex/2008/moorings/m08/pt/rbr_011855/raw/pt_rbr_011855.dat")
d <- subset(rbr011855, time < as.POSIXct("2008-06-25 10:05:00"))
x <- undriftTime(d, 1)   # clock lost 1 second over whole experiment
summary(d)
summary(x)

## End(Not run)
```

---

unduplicateNames *Rename duplicated items (used in reading CTD files)*

Description

Rename items to avoid name collision, by appending a 2 to the second occurrence of a name, etc.

Usage

```
unduplicateNames(names)
```

Arguments

names	Vector of strings with variable names.

Value

names Vector of strings with numbered variable names.

See Also

used by `read.ctd.sbe`.

Examples

```
unduplicateNames(c("a", "b", "a", "c", "b"))
```

---

ungrid — *Extract (x, y, z) from (x, y, grid)*

Description

Extract the grid points from a grid, returning columns. This is useful for e.g. gridding large datasets, in which the first step might be to use `binMean2D`, followed by `interpBarnes`.

Usage

```
ungrid(x, y, grid)
```

Arguments

`x`	a vector holding the x coordinates of grid.
`y`	a vector holding the y coordinates of grid.
`grid`	a matrix holding the grid.

Value

A list containing three vectors: `x`, the grid x values, `y`, the grid y values, and `grid`, the grid values.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(wind)
u <- interpBarnes(wind$x, wind$y, wind$z)
contour(u$xg, u$yg, u$zg)
U <- ungrid(u$xg, u$yg, u$zg)
points(U$x, U$y, col=oce.colorsJet(100)[rescale(U$grid, rlow=1, rhigh=100)], pch=20)
```

unitFromString *Decode units, from strings*

Description

Decode units, from strings

Usage

```
unitFromString(s)
```

Arguments

s	A string.

Value

A list of two items: unit which is an expression, and scale, which is a string.

See Also

Other functions that interpret variable names and units from headers: ODFNames2oceNames, cnvName2oceName, oceNames2whpNames, oceUnits2whpUnits, unitFromStringRsk, woceNames2oceNames, woceUnit2oceUnit

Examples

```
unitFromString("DB") # dbar
```

unitFromStringRsk *Infer Rsk units from a vector of strings*

Description

This is used by read.rsk to infer the units of data, based on strings stored in .rsk files. Lacking a definitive guide to the format of these file, this function was based on visual inspection of the data contained within a few sample files; unusual sensors may not be handled properly.

Usage

```
unitFromStringRsk(s)
```

Arguments

s	Vector of character strings, holding the ‘units‘ entry in the channels table of the .rsk database.

Value

List of unit lists.

See Also

Other functions that interpret variable names and units from headers: ODFNames2oceNames, cnvName2oceName, oceNames2whpNames, oceUnits2whpUnits, unitFromString, woceNames2oceNames, woceUnit2oceUnit

---

unwrapAngle *Unwrap an angle that suffers modulo-360 problems*

Description

This is mostly used for instrument heading angles, in cases where the instrument is aligned nearly northward, so that small variations in heading (e.g. due to mooring motion) can yield values that swing from small angles to large angles, because of the modulo-360 cut point. The method is to use the cosine and sine of the angle, to construct "x" and "y" values on a unit circle, then to find means and medians of x and y respectively, and finally to use atan2 to infer the angles.

Usage

```
unwrapAngle(angle)
```

Arguments

angle	an angle (in degrees) that is thought be near 360 degrees, with added noise

Value

A list with two estimates: `mean` is based on an arithmetic mean, and `median` is based on the median. Both are mapped to the range 0 to 360.

Author(s)

Dan Kelley

Examples

```
library(oce)
true <- 355
a <- true + rnorm(100, sd=10)
a <- ifelse(a > 360, a - 360, a)
a2 <- unwrapAngle(a)
par(mar=c(3, 3, 5, 3))
hist(a, breaks=360)
abline(v=a2$mean, col="blue", lty="dashed")
abline(v=true, col="blue")
```

```
mtext("true (solid)\n estimate (dashed)", at=true, side=3, col="blue")
abline(v=mean(a), col="red")
mtext("mean", at=mean(a), side=3, col="red")
```

---

useHeading *Replace the Heading for One Instrument With That of Another*

Description

Replace the heading angles in one oce object with that from another, possibly with a constant adjustment.

Usage

```
useHeading(b, g, add = 0)
```

Arguments

b	object holding data from an instrument whose heading is bad, but whose other data are good.
g	object holding data from an instrument whose heading is good, and should be interpolated to the time base of b.
add	an angle, in degrees, to be added to the heading.

Value

A copy of b, but with b$data$heading replaced with heading angles that result from linear interpolation of the headings in g, and then adding the angle add.

Author(s)

Dan Kelley

---

usrLonLat *Calculate lon-lat coordinates of plot-box trace*

Description

Trace along the plot box, converting from xy coordinates to lonlat coordinates. The results are used by mapGrid and mapAxis to ignore out-of-frame grid lines and axis labels.

Usage

```
usrLonLat(n = 25, debug = getOption("oceDebug"))
```

Arguments

n	number of points to check along each side of the plot box
debug	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

Details

Note: this procedure does not work for projections that have trouble inverting points that are "off the globe". For this reason, this function examines .Projection()$projection and if it contains the string "wintri", then the above-stated procedure is skipped, and the return value has each of the numerical quantities set to NA, and ok set to FALSE.

Value

a list containing lonmin, lonmax, latmin, latmax, and ok; the last of which indicates whether at least one point on the plot box is invertible. Note that longitude are expressed in the range from -180 to 180 degrees.

Author(s)

Dan Kelley

See Also

Other functions related to maps: lonlat2map, lonlat2utm, map2lonlat, mapArrows, mapAxis, mapContour, mapDirectionField, mapGrid, mapImage, mapLines, mapLocator, mapLongitudeLatitudeXY, mapPlot, mapPoints, mapPolygon, mapScalebar, mapText, mapTissot, oceCRS, shiftLongitude, utm2lonlat

---

utm2lonlat *Convert UTM to Longitude and Latitude*

Description

Convert UTM to Longitude and Latitude

Usage

```
utm2lonlat(easting, northing, zone = 1, hemisphere = "N", km = FALSE)
```

Arguments

`easting`	easting coordinate (in km or m, depending on value of `km`). Alternatively, a list containing items named `easting`, `northing`, and `zone`, in which case these are taken from the list and the arguments named `northing`, `zone` and are ignored.
`northing`	northing coordinate (in km or m, depending on value of `km`).
`zone`	UTM zone
`hemisphere`	indication of hemisphere; `"N"` for North, anything else for South.
`km`	logical value indicating whether `easting` and `northing` are in kilometers or meters.

Value

A list containing `longitude` and `latitude`.

Author(s)

Dan Kelley

References

http://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system, downloaded May 31, 2014.

See Also

`lonlat2utm` does the inverse operation. For general projections and their inverses, use `lonlat2map` and `map2lonlat`.

Other functions related to maps: `lonlat2map`, `lonlat2utm`, `map2lonlat`, `mapArrows`, `mapAxis`, `mapContour`, `mapDirectionField`, `mapGrid`, `mapImage`, `mapLines`, `mapLocator`, `mapLongitudeLatitudeXY`, `mapPlot`, `mapPoints`, `mapPolygon`, `mapScalebar`, `mapText`, `mapTissot`, `oceCRS`, `shiftLongitude`, `usrLonLat`

Examples

```
library(oce)
## Cape Split, in the Minas Basin of the Bay of Fundy
utm2lonlat(852863, 5029997, 19)
```

vectorShow — *Show some values from a vector*

Description

This is similar to `str`, but it shows data at the first and last of the vector, which can be quite helpful in debugging.

Usage

```
vectorShow(v, msg, digits = 5, n = 2L)
```

Arguments

`v`	the vector.
`msg`	a message to show, introducing the vector. If not provided, then a message is created from `v`.
`digits`	for numerical values of `v`, this is the number of digits to use, in formatting the numbers with `format`; otherwise, `digits` is ignored.
`n`	number of elements to at start and end. If `n` is negative, then all the elements are shown.

Value

A string, suitable for using in `cat` or `oceDebug`.

Author(s)

Dan Kelley

velocityStatistics — *Report Statistics of adp or adv Velocities*

Description

Report statistics of ADP or ADV velocities, such as means and variance ellipses.

Usage

```
velocityStatistics(x, control, ...)
```

Arguments

x	an adp or adv object, i.e. one inheriting from adp-class or adv-class.
control	An optional list used to specify more information. This is presently ignored for adv objects. For adp objects, if control$bin is an integer, it is taken as the bin to be selected (otherwise, an average across bins is used).
...	additional arguments that are used in the call to mean.

Value

A list containing items the major and minor axes of the covariance ellipse (ellipseMajor and ellipseMinor), the angle of the major axis anticlockwise of the horizontal axis (ellipseAngle), and the x and y components of the mean velocity (uMean and vMean).

Author(s)

Dan Kelley

See Also

Other things related to adp data: [[,adp-method, [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, xyzToEnuAdp, xyzToEnu

Other things related to adv data: [[,adv-method, [[<-,adv-method, adv-class, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, xyzToEnuAdv, xyzToEnu

Examples

```
library(oce)
data(adp)
a <- velocityStatistics(adp)
print(a)
t <- seq(0, 2*pi, length.out=100)
theta <- a$ellipseAngle * pi / 180
y <- a$ellipseMajor * cos(t) * sin(theta) + a$ellipseMinor * sin(t) * cos(theta)
x <- a$ellipseMajor * cos(t) * cos(theta) - a$ellipseMinor * sin(t) * sin(theta)
plot(adp, which="uv+ellipse+arrow")
lines(x, y, col='blue', lty="dashed", lwd=5)
arrows(0, 0, a$uMean, a$vMean, lwd=5, length=1/10, col='blue', lty="dashed")
```

webtide *Get a Tidal Prediction from a WebTide Database*

Description

Get a tidal prediction from a WebTide database. There are two distinct cases.

Case 1: `action="map"`. In this case, if `plot` is `FALSE`, a list is returned, containing all the `node`s in the selected database, along with all the `latitude`s and `longitude`s. This value is also returned (silently) if `plot` is true, but in that case, a plot is drawn to indicate the node locations. If `latitude` and `longitude` are given, then the node nearest that spot is indicated on the map; otherwise, if `node` is given, then the location of that node is indicated. There is also a special case: if `node` is negative and `interactive()` is `TRUE`, then `locator` is called, and the node nearest the spot where the user clicks the mouse is indicated in the plot and in the return value.

`Case 2: action="predict"`. If `plot` is `FALSE`, then a list is returned, indicating `time`, predicted `elevation`, velocity components `u` and `v`, `node` number, the name of the `basedir`, and the `region`. If `plot` is `TRUE`, this list is returned silently, and time-series plots are drawn for elevation, u, and v.

Naturally, `webtide` will not work unless WebTide has been installed on the computer.

Usage

```
webtide(action = c("map", "predict"), longitude, latitude, node, time,
  basedir = getOption("webtide"), region = "nwatl", plot = TRUE, tformat,
  debug = getOption("oceDebug"), ...)
```

Arguments

`action`	An indication of the action, either `action="map"` to draw a map or `action="predict"` to get a prediction; see 'Details'.
`longitude, latitude`	optional location at which prediction is required (ignored if `node` is given).
`node`	optional integer relating to a node in the database. If `node` is given, then neither `latitude` nor `longitude` may be given. If `node` is positive, then specifies indicates the node. If it is negative, `locator` is called so that the user can click (once) on the map, after which the node is displayed on the map.
`time`	a vector of times at which prediction is to be made. If not supplied, this will be the week starting at the present time, incrementing by 15 minutes.
`basedir`	directory containing the `WebTide` application.
`region`	database region, given as a directory name in the WebTide directory. For example, `h3o` is for Halifax Harbour, `nwatl` is for the northwest Atlantic, and `sshelf` is for the Scotian Shelf and Gulf of Maine.
`plot`	boolean indicating whether to plot.
`tformat`	optional argument passed to `oce.plot.ts`, for plot types that call that function. (See `strptime` for the format used.)

debug
: an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many oce functions. Generally, setting debug=0 turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of debug first, so that a user can often obtain deeper debugging by specifying higher debug values.

...
: optional arguments passed to plotting functions. A common example is to set xlim and ylim, to focus a map region.

Value

If action="map" the return value is a list containing the index of the nearest node, along with the latitude and longitude of that node. If plot is FALSE, this value is returned invisibly.

If action="predict", the return value is a list containing a vector of times (time), as well as vectors of the predicted elevation in metres and the predicted horizontal components of velocity, u and v, along with the node number, and the basedir and region as supplied to this function. If plot is FALSE, this value is returned invisibly.

a list containing node, longitude and latitude. If no node was provided to webtide then this list will cover the whole set of nodes in the WebTide model; otherwise, it will be just the node that was requested.

Caution

WebTide is not an open-source application, so the present function was designed based on little more than guesses about the WebTide file structure. Users should be on the lookout for odd results.

Author(s)

Dan Kelley

Source

The WebTide software may be downloaded for free at the Department of Fisheries and Oceans (Canada) website at http://www.bio.gc.ca/science/research-recherche/ocean/webtide/index-en.php (checked February 2016 and May 2017).

Examples

```
## Not run:
## needs WebTide at the system level
library(oce)
## 1. prediction at Halifax NS
longitude <- -63.57
latitude <- 44.65
prediction <- webtide("predict", longitude=longitude, latitude=latitude)
mtext(sprintf("prediction at %fN %fE", latitude, longitude), line=0.75, side=3)
## 2. map
webtide(lon=-63.57,lat=44.65,xlim=c(-64,-63),ylim=c(43.0,46))

## End(Not run)
```

`wind` *Wind dataset*

Description

Wind data inferred from Figure 5 of Koch et al. (1983), provided to illustrate the `interpBarnes` function. Columns `wind$x` and `wind$y` are location, while `wind$z` is the wind speed, in m/s.

References

S. E. Koch and M. DesJardins and P. J. Kocin, 1983. "An interactive Barnes objective map analysis scheme for use with satellite and conventional data," *J. Climate Appl. Met.*, vol 22, p. 1487-1503.

See Also

Other datasets provided with oce: `adp`, `adv`, `argo`, `cm`, `coastlineWorld`, `colors`, `ctdRaw`, `ctd`, `echosounder`, `landsat`, `lisst`, `lobo`, `met`, `rsk`, `sealevelTuktoyaktuk`, `sealevel`, `section`, `topoWorld`

`window.oce` *Window an Oce Object by Time or Distance*

Description

Windows `x` on either time or distance, depending on the value of `which`. In each case, values of `start` and `end` may be integers, to indicate a portion of the time or distance range. If `which` is `"time"`, then the `start` and `end` values may also be provided as POSIX times, or character strings indicating times (in time zone given by the value of `getOption("oceTz")`). Note that `subset` may be more useful than this function.

Usage

```
## S3 method for class 'oce'
window(x, start = NULL, end = NULL, frequency = NULL,
  deltat = NULL, extend = FALSE, which = c("time", "distance"),
  indexReturn = FALSE, debug = getOption("oceDebug"), ...)
```

Arguments

`x`	an oce object.
`start`	the start time (or distance) of the time (or space) region of interest. This may be a single value or a vector.
`end`	the end time (or distance) of the time (or space) region of interest. This may be a single value or a vector.
`frequency`	not permitted yet.

`deltat`	not permitted yet
`extend`	not permitted yet
`which`	string containing the name of the quantity on which sampling is done. Possibilities are `"time"`, which applies the windowing on the `time` entry of the `data` slot, and `"distance"`, for the `distance` entry (for those objects, such as `adp`, that have this entry).
`indexReturn`	boolean flag indicating whether to return a list of the "kept" indices for the `time` entry of the `data` slot, as well as the `timeSlow` entry, if there is one.. Either of these lists will be `NULL`, if the object lacks the relevant items.
`debug`	a flag that turns on debugging.
`...`	ignored

Value

Normally, this is new `oce` object. However, if `indexReturn=TRUE`, the return value is two-element list containing items named `index` and `indexSlow`, which are the indices for the `time` entry of the `data` slot (and the `timeSlow`, if it exists).

Author(s)

Dan Kelley

See Also

`subset` provides more flexible selection of subsets.

Examples

```
library(oce)
data(adp)
plot(adp)
early <- window(adp, start="2008-06-26 00:00:00", end="2008-06-26 12:00:00")
plot(early)
bottom <- window(adp, start=0, end=20, which="distance")
plot(bottom)
```

`windrose-class` *Class to Store Windrose Data*

Description

Windrose objects store statistical information about winds, mainly for plotting as "wind rose" plots (see `plot,windrose-method`. There is no reading method, because there is no standard way to store wind data in files; instead, `as.windrose` is provided to construct `windrose` objects. Data elements may be retrieved with `[[,windrose-method` or replaced with `[[<-,windrose-method`. Data summaries are provided with `summary,windrose-method`.

See Also

Other classes provided by oce: adp-class, adv-class, argo-class, bremen-class, cm-class, coastline-class, ctd-class, echosounder-class, lisst-class, lobo-class, met-class, oce-class, odf-class, rsk-class, sealevel-class, section-class, topo-class

Other things related to windrose data: [[,windrose-method, [[<-,windrose-method, as.windrose, plot,windrose-method, summary,windrose-method

---

woceNames2oceNames *Translate WOCE Data Names to Oce Data Names*

Description

Translate WOCE-style names to oce names, using gsub to match patterns. For example, the pattern "CTDOXY.*" is taken to mean oxygen.

Usage

```
woceNames2oceNames(names)
```

Arguments

names	vector of strings holding WOCE-style names.

Value

vector of strings holding oce-style names.

Author(s)

Dan Kelley

References

Several online sources list WOCE names. An example is https://geo.h2o.ucsd.edu/documentation/manuals/pdf/90_1/chap4.pdf

See Also

Other things related to ctd data: [[,ctd-method, [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceUnit2oceUnit, write.ctd

Other functions that interpret variable names and units from headers: ODFNames2oceNames, cnvName2oceName, oceNames2whpNames, oceUnits2whpUnits, unitFromStringRsk, unitFromString, woceUnit2oceUnit

woceUnit2oceUnit *Translate WOCE units to oce units*

Description

Translate WOCE-style units to oce units.

Usage

```
woceUnit2oceUnit(woceUnit)
```

Arguments

`woceUnit`	string holding a WOCE unit

Value

expression in oce unit form

Author(s)

Dan Kelley

See Also

Other things related to ctd data: `[[,ctd-method`, `[[<-,ctd-method`, `as.ctd`, `cnvName2oceName`, `ctd-class`, `ctdDecimate`, `ctdFindProfiles`, `ctdRaw`, `ctdTrim`, `ctd`, `handleFlags,ctd-method`, `oceNames2whpNames`, `oceUnits2whpUnits`, `plot,ctd-method`, `plotProfile`, `plotScan`, `plotTS`, `read.ctd.itp`, `read.ctd.odf`, `read.ctd.sbe`, `read.ctd.woce.other`, `read.ctd.woce`, `read.ctd`, `subset,ctd-method`, `summary,ctd-method`, `woceNames2oceNames`, `write.ctd`

Other functions that interpret variable names and units from headers: `ODFNames2oceNames`, `cnvName2oceName`, `oceNames2whpNames`, `oceUnits2whpUnits`, `unitFromStringRsk`, `unitFromString`, `woceNames2oceNames`

write.ctd *Write a CTD Data Object as a CSV File*

Description

Writes a comma-separated file containing the data frame stored in the `data` slot of the first argument. The file is suitable for reading with a spreadsheet, or with `read.csv`. This output file will contain some of the metadata in `x`, if `metadata` is `TRUE`.

Usage

```
write.ctd(object, file, metadata = TRUE, flags = TRUE, format = "csv")
```

Arguments

`object`	A `ctd` object, i.e. one inheriting from `ctd-class`.
`file`	Either a character string (the file name) or a connection. If not provided, `file` defaults to `stdout()`.
`metadata`	a logical value indicating whether to put some selected metadata elements at the start of the output file.
`flags`	a logical value indicating whether to show data-quality flags as well as data.
`format`	string indicating the format to use. This may be `"csv"` for a simple CSV format, or `"whp"` for the World Hydrographic Program format, described at [1] and exemplified at [2].

Author(s)

Dan Kelley

References

1. https://www.nodc.noaa.gov/woce/woce_v3/wocedata_1/whp/exchange/exchange_format_desc.htm
2. https://www.nodc.noaa.gov/woce/woce_v3/wocedata_1/whp/exchange/example_ct1.csv

See Also

The documentation for `ctd-class` explains the structure of CTD objects.

Other things related to `ctd` data: `[[,ctd-method`, `[[<-,ctd-method`, `as.ctd`, `cnvName2oceName`, `ctd-class`, `ctdDecimate`, `ctdFindProfiles`, `ctdRaw`, `ctdTrim`, `ctd`, `handleFlags,ctd-method`, `oceNames2whpNames`, `oceUnits2whpUnits`, `plot,ctd-method`, `plotProfile`, `plotScan`, `plotTS`, `read.ctd.itp`, `read.ctd.odf`, `read.ctd.sbe`, `read.ctd.woce.other`, `read.ctd.woce`, `read.ctd`, `subset,ctd-method`, `summary,ctd-method`, `woceNames2oceNames`, `woceUnit2oceUnit`

Examples

```
## Not run:
library(oce)
data(ctd)
write.ctd(ctd, "ctd.csv")
d <- read.csv("ctd.csv")
plot(as.ctd(d$salinity, d$temperature, d$pressure))

## End(Not run)
```

xyzToEnu — *Convert acoustic-Doppler data from xyz coordinates to enu coordinates*

Description

Convert acoustic-Doppler data from xyz coordinates to enu coordinates

Usage

```
xyzToEnu(x, ...)
```

Arguments

`x`	an adp or adv object, i.e. one inheriting from `adp-class` or `adv-class`.
`...`	extra arguments that are passed on to `xyzToEnuAdp` or `xyzToEnuAdv`.

Value

An object of the same type as `x`, but with velocities in east-north-up coordinates instead of xyz coordinates.

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnuAdp`

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOtherAdv`, `enuToOther`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnuAdv`

xyzToEnuAdp — *Convert ADP From XYZ to ENU Coordinates*

Description

Convert ADP velocity components from a xyz-based coordinate system to an enu-based coordinate system, by using the instrument's recording of heading, pitch, and roll.

Usage

```
xyzToEnuAdp(x, declination = 0, debug = getOption("oceDebug"))
```

Arguments

`x`	An `adp` object, i.e. one inheriting from `adp-class`.
`declination`	magnetic declination to be added to the heading after "righting" (see below), to get ENU with N as "true" north.
`debug`	an integer specifying whether debugging information is to be printed during the processing. This is a general parameter that is used by many `oce` functions. Generally, setting `debug=0` turns off the printing, while higher values suggest that more information be printed. If one function calls another, it usually reduces the value of `debug` first, so that a user can often obtain deeper debugging by specifying higher `debug` values.

Details

The first step is to convert the (x,y,z) velocity components (stored in the three columns of `x[["v"]][,,1:3]`) into what RDI [1, pages 11 and 12] calls "ship" (or "righted") components. For example, the z coordinate, which may point upwards or downwards depending on instrument orientation, is mapped onto a "mast" coordinate that points more nearly upwards than downward. The other ship coordinates are called "starboard" and "forward", the meanings of which will be clear to mariners. Once the (x,y,z) velocities are converted to ship velocities, the orientation of the instrument is extracted from heading, pitch, and roll vectors stored in the object. These angles are defined differently for RDI and Sontek profilers.

The code handles every case individually, based on the table given below. The table comes from Clark Richards, a former PhD student at Dalhousie University [2], who developed it based on instrument documentation, discussion on user groups, and analysis of measurements acquired with RDI and Sontek acoustic current profilers in the SLEIWEX experiment. In the table, (X, Y, Z) denote instrument-coordinate velocities, (S, F, M) denote ship-coordinate velocities, and (H, P, R) denote heading, pitch, and roll.

Case	**Mfr.**	**Instr. Orient.**	**H**	**P**	**R**	**S**	**F**	**M**	
1	RDI	ADCP	up	H	arctan(tan(P)*cos(R))	R	-X	Y	-Z
2	RDI	ADCP	down	H	arctan(tan(P)*cos(R))	-R	X	Y	Z
3	Nortek	ADP	up	H-90	R	-P	X	Y	Z
4	Nortek	ADP	down	H-90	R	-P	X	-Y	-Z
5	Sontek	ADP	up	H-90	-P	-R	X	Y	Z
6	Sontek	ADP	down	H-90	-P	-R	X	Y	Z
7	Sontek	PCADP	up	H-90	R	-P	X	Y	Z
8	Sontek	PCADP	down	H-90	R	-P	X	Y	Z

Finally, a standardized rotation matrix is used to convert from ship coordinates to earth coordinates. As described in the RDI coordinate transformation manual [1, pages 13 and 14], this matrix is based on sines and cosines of heading, pitch, and roll If `CH` and `SH` denote cosine and sine of heading (after adjusting for declination), with similar terms for pitch and roll using second letters `P` and `R`, the rotation matrix is

```
 rbind(c( CH*CR + SH*SP*SR, SH*CP, CH*SR - SH*SP*CR), c(-SH*CR
+ CH*SP*SR, CH*CP, -SH*SR - CH*SP*CR), c( -CP*SR, SP, CP*CR))
```

This matrix is left-multiplied by a matrix with three rows, the top a vector of "starboard" values, the middle a vector of "forward" values, and the bottom a vector of "mast" values. Finally, the columns of `data$v[,,1:3]` are filled in with the result of the matrix multiplication.

Value

An object with `data$v[,,1:3]` altered appropriately, and `metadata$oce.orientation` changed from `xyz` to `enu`.

Author(s)

Dan Kelley and Clark Richards

References

1. RD Instruments, 1998. *ADCP Coordinate Transformation, formulas and calculations.* P/N 951-6079-00 (July 1998).

2. Clark Richards, 2012, PhD Dalhousie University Department of Oceanography.

See Also

Other things related to adp data: `[[,adp-method`, `[[<-,adp-method`, `adp-class`, `adpEnsembleAverage`, `adp`, `as.adp`, `beamName`, `beamToXyzAdp`, `beamToXyzAdv`, `beamToXyz`, `beamUnspreadAdp`, `binmapAdp`, `enuToOtherAdp`, `enuToOther`, `plot,adp-method`, `read.ad2cp`, `read.adp.nortek`, `read.adp.rdi`, `read.adp.sontek.serial`, `read.adp.sontek`, `read.adp`, `read.aquadoppHR`, `read.aquadoppProfiler`, `read.aquadopp`, `subset,adp-method`, `summary,adp-method`, `toEnuAdp`, `toEnu`, `velocityStatistics`, `xyzToEnu`

`xyzToEnuAdv` *Convert an ADP from XYZ to ENU Coordinates*

Description

Convert ADV velocity components from a xyz-based coordinate system to an enu-based coordinate system.

Usage

```
xyzToEnuAdv(x, declination = 0, cabled = FALSE, horizontalCase,
  sensorOrientation, debug = getOption("oceDebug"))
```

Arguments

x	An adv object, i.e. one inheriting from adv-class.
declination	magnetic declination to be added to the heading, to get ENU with N as "true" north.
cabled	boolean value indicating whether the sensor head is connected to the pressure case with a cable. If cabled=FALSE, then horizontalCase is ignored. See “Details”.
horizontalCase	optional boolean value indicating whether the sensor case is oriented horizontally. Ignored unless cabled is TRUE. See “Details”.
sensorOrientation	optional string indicating the direction in which the sensor points. The value, which must be "upward" or "downward", over-rides the value of orientation, in the metadata slot, which will have been set by read.adv, *provided* that the data file contained the full header. See “Details”.
debug	a flag that, if non-zero, turns on debugging. Higher values yield more extensive debugging.

Details

The coordinate transformation is done using the heading, pitch, and roll information contained within x. The algorithm is similar to that used for Teledyne/RDI ADCP units, taking into account the different definitions of heading, pitch, and roll as they are defined for the velocimeters.

Generally, the transformation must be done on a time-by-time basis, which is a slow operation. However, this function checks whether the vectors for heading, pitch and roll, are all of unit length, and in that case, the calculation is altered, resulting in shorter execution times. Note that the angles are held in (data$timeSlow, data$headingSlow, ...) for Nortek instruments and (data$time, data$heading, ...) for Sontek instruments.

Since the documentation provided by instrument manufacturers can be vague on the coordinate transformations, the method used here had to be developed indirectly. (This is in contrast to the RDI ADCP instruments, for which there are clear instructions.) documents that manufacturers provide. If results seem incorrect (e.g. if currents go east instead of west), users should examine the code in detail for the case at hand. The first step is to set debug to 1, so that the processing will print a trail of processing steps. The next step should be to consult the table below, to see if it matches the understanding (or empirical tests) of the user. It should not be difficult to tailor the code, if needed.

The code handles every case individually, based on the table given below. The table comes from Clark Richards, a former PhD student at Dalhousie University [2], who developed it based on instrument documentation, discussion on user groups, and analysis of measurements acquired with Nortek and Sontek velocimeters in the SLEIWEX experiment.

The column labelled “Cabled” indicates whether the sensor and the pressure case are connected with a flexible cable, and the column labelled “H.case” indicates whether the pressure case is oriented horizontally. These two properties are not discoverable in the headers of the data files, and so they must be supplied with the arguments cabled and horizontalCase. The source code refers to the information in this table by case numbers. (Cases 5 and 6 are not handled.) Angles are abbreviated as follows:: heading “H,” pitch “P,” and roll “R”. Entries X, Y and Z refer to instrument coordinates of the same names. Entries S, F and M refer to so-called ship coordinates starboard, forward, and mast; it is these that are used together with a rotation matrix to get velocity components in the east, north, and upward directions.

Case	Mfr.	Instr.	Cabled	H. case	Orient.	H	P	R	S	F	M
1	Nortek	vector	no	-	up	H-90	R	-P	X	-Y	-Z
2	Nortek	vector	no	-	down	H-90	R	-P	X	Y	Z
3	Nortek	vector	yes	yes	up	H-90	R	-P	X	Y	Z
4	Nortek	vector	yes	yes	down	H-90	R	P	X	-Y	-Z
5	Nortek	vector	yes	no	up	-	-	-	-	-	-
6	Nortek	vector	yes	no	down	-	-	-	-	-	-
7	Sontek	adv	-	-	up	H-90	R	-P	X	-Y	-Z
8	Sontek	adv	-	-	down	H-90	R	-P	X	Y	Z

Author(s)

Dan Kelley, in collaboration with Clark Richards

References

1. http://www.nortek-as.com/lib/forum-attachments/coordinate-transformation

2. Clark Richards, 2012, PhD Dalhousie University Department of Oceanography.

See Also

See `read.adv` for notes on functions relating to adv objects.

Other things related to adv data: `[[,adv-method`, `[[<-,adv-method`, `adv-class`, `adv`, `beamName`, `beamToXyz`, `enuToOtherAdv`, `enuToOther`, `plot,adv-method`, `read.adv.nortek`, `read.adv.sontek.adr`, `read.adv.sontek.serial`, `read.adv.sontek.text`, `read.adv`, `subset,adv-method`, `summary,adv-method`, `toEnuAdv`, `toEnu`, `velocityStatistics`, `xyzToEnu`

`[[,adp-method` *Extract Parts of an ADP Object In addition to the usual extraction of elements by name, some shortcuts are also provided, e.g.* `x[["u1"]]` *retrieves* `v[,1]`, *and similarly for the other velocity components. The* `a` *and* `q` *data can be retrieved in* `raw` *form or numeric form (see examples). The coordinate system may be retrieved with e.g.* `x[["coordinate"]]`.

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ...

method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, NULL is returned.

Usage

```
## S4 method for signature 'adp'
x[[i, j, ...]]
```

Arguments

x	An adp object, i.e. one inheriting from adp-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of i.

First, a check is made as to whether i names one of the standard oce slots, and returns the slot contents if so. Thus, x[["metadata"]] will retrieve the metadata slot, while x[["data"]] and x[["processingLog"]] return those slots.

Next, if i is a string ending in the "Unit", then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named unit, which is an expression, and an item named scale, which is a string describing the measurement scale. If the string ends in " unit", e.g. x[["temperature unit"]], then just the expression is returned, and if it ends in " scale", then just the scale is returned.

Next, if i is a string ending in "Flag", then the corresponding data-quality flag is returned (or NULL if there is no such flag). For example, x[["salinityFlag"]] returns a vector of salinity flags if x is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

Author(s)

Dan Kelley

See Also

Other functions that extract parts of oce objects: [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method,

[[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to adp data: [[<-,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

Examples

```
data(adp)
# Tests for beam 1, distance bin 1, first 5 observation times
adp[["v"]][1:5,1,1]
adp[["a"]][1:5,1,1]
adp[["a", "numeric"]][1:5,1,1]
as.numeric(adp[["a"]][1:5,1,1]) # same as above
```

[[,adv-method *Extract Parts of an ADV Object*

Description

The [[method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related replacement method, [[<-, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, NULL is returned.

Usage

```
## S4 method for signature 'adv'
x[[i, j, ...]]
```

Arguments

x	An adv object, i.e. one inheriting from adv-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the specialized adv method

In addition to the usual extraction of elements by name, some shortcuts are also provided, e.g. u1 retrieves v[,1], and similarly for the other velocity components. The a and q data can be retrieved in raw form or numeric form; see "Examples".

It is also worth noting that heading, pitch, etc. may be stored in "slow" form in the object (e.g. in headingSlow within the data slot). In that case, accessing by full name, e.g. x[["headingSlow"]] retrieves the item as expected, but x[["heading"]] interpolates to the faster timescale, using approx(timeSlow, headingSlow, time).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of i.

First, a check is made as to whether i names one of the standard oce slots, and returns the slot contents if so. Thus, x[["metadata"]] will retrieve the metadata slot, while x[["data"]] and x[["processingLog"]] return those slots.

Next, if i is a string ending in the "Unit", then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named unit, which is an expression, and an item named scale, which is a string describing the measurement scale. If the string ends in " unit", e.g. x[["temperature unit"]], then just the expression is returned, and if it ends in " scale", then just the scale is returned.

Next, if i is a string ending in "Flag", then the corresponding data-quality flag is returned (or NULL if there is no such flag). For example, x[["salinityFlag"]] returns a vector of salinity flags if x is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to adv data: [[<-,adv-method, adv-class, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

Examples

```
data(adv)
head(adv[["q"]])            # in raw form
head(adv[["q", "numeric"]]) # in numeric form
```

`[[,amsr-method` *Extract Something From an amsr Object*

Description

Extract something from the `metadata` or `data` slot of an `amsr-class` object.

The `[[` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'amsr'
x[[i, j, ...]]
```

Arguments

`x`	An `amsr` object, i.e. one inheriting from `amsr-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details

Partial matches for `i` are permitted for `metadata`, and `j` is ignored for `metadata`.

Data within the `data` slot may be found directly, e.g. `j="SSTDay"` will yield sea-surface temperature in the daytime satellite, and `j="SSTNight"` is used to access the nighttime data. In addition, `j="SST"` yields an average of the night and day values (using just one of these, if the other is missing). This scheme works for all the data stored in `amsr` objects, namely: `time`, `SST`, `LFwind`, `MFwind`, `vapor`, `cloud` and `rain`. In each case, the default is to calculate values in scientific units, unless `j="raw"`, in which case the raw data are returned.

The `"raw"` mode can be useful in decoding the various types of missing value that are used by `amsr` data, namely `as.raw(255)` for land, `as.raw(254)` for a missing observation, `as.raw(253)` for a bad observation, `as.raw(252)` for sea ice, or `as.raw(251)` for missing SST due to rain or missing water vapour due to heavy rain. Note that something special has to be done for e.g. `d[["SST", "raw"]]` because the idea is that this syntax (as opposed to specifying `"SSTDay"`) is a request to try to find good data by looking at both the Day and Night measurements. The scheme employed is quite detailed. Denote by "A" the raw value of the desired field in the daytime pass, and by "B" the corresponding value in the nighttime pass. If either A or B is 255, the code for land, then the result will be 255. If A is 254 (i.e. there is no observation), then B is returned, and the reverse holds also. Similarly, if either A or B equals 253 (bad observation), then the other is returned. The same is done for code 252 (ice) and code 251 (rain).

Value

In all cases, the returned value is a matrix with with `NA` values inserted at locations where the raw data equal `as.raw(251:255)`, as explained in "Details".

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

Author(s)

Dan Kelley

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Examples

```
## Not run:
# Show a daytime SST image, along with an indication of whether
# the NA values are from rain.
library(oce)
earth <- read.amsr("f34_20160102v7.2.gz")
fclat <- subset(earth , 35 <= latitude & latitude <= 55)
fc <- subset(fclat , -70 <= longitude & longitude <= -30)
par(mfrow=c(2, 1))
plot(fc, "SSTDay")
rainy <- fc[["SSTDay", "raw"]] == as.raw(0xfb)
lon <- fc[["longitude"]]
lat <- fc[["latitude"]]
asp <- 1 / cos(pi*mean(lat)/180)
imagep(lon, lat, rainy, asp=asp)
mtext("red: too rainy to sense SSTDay")

## End(Not run)
```

`[[,argo-method` *Extract Something From an Argo Object*

Description

The `[[` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'argo'
x[[i, j, ...]]
```

Arguments

`x`	An `argo` object, i.e. one inheriting from `argo-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the specialized argo method

- If `i` is the string `"SA"`, then the method computes Absolute Salinity using `gsw_SA_from_SP`, using `salinityAdjusted` (etc) if available in the `data` slot of `x`, otherwise using `salinity`.
- Similarly, for `"CT"`, Conservative Temperature is returned.
- Otherwise, if `i` is in the `data` slot of `x`, then it is returned, otherwise if it is in the `metadata` slot, then that is returned, otherwise `NULL` is returned.

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Other things related to argo data: `[[<-,argo-method`, `argo-class`, `argoGrid`, `argoNames2oceNames`, `argo`, `as.argo`, `handleFlags,argo-method`, `plot,argo-method`, `read.argo`, `subset,argo-method`, `summary,argo-method`

Examples

```
data(argo)
dim(argo[['temperature']])
```

`[[,bremen-method` *Extract Something From a Bremen Object*

Description

The `[[` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'bremen'
x[[i, j, ...]]
```

Arguments

`x`	A bremen object, i.e. one inheriting from `bremen-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to bremen data: [[<-,bremen-method, bremen-class, plot,bremen-method, read.bremen, summary,bremen-method

---

[[,cm-method *Extract Something From a CM Object*

Description

The [[method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related replacement method, [[<-, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, NULL is returned.

Usage

```
## S4 method for signature 'cm'
x[[i, j, ...]]
```

Arguments

x	A cm object, i.e. one inheriting from cm-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Other things related to `cm` data: `[[<-,cm-method`, `as.cm`, `cm-class`, `cm`, `plot,cm-method`, `read.cm`, `subset,cm-method`, `summary,cm-method`

`[[,coastline-method` *Extract Something From a Coastline Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'coastline'
x[[i, j, ...]]
```

Arguments

x	A coastline object, i.e. one inheriting from coastline-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of i.

First, a check is made as to whether i names one of the standard oce slots, and returns the slot contents if so. Thus, x[["metadata"]] will retrieve the metadata slot, while x[["data"]] and x[["processingLog"]] return those slots.

Next, if i is a string ending in the "Unit", then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named unit, which is an expression, and an item named scale, which is a string describing the measurement scale. If the string ends in " unit", e.g. x[["temperature unit"]], then just the expression is returned, and if it ends in " scale", then just the scale is returned.

Next, if i is a string ending in "Flag", then the corresponding data-quality flag is returned (or NULL if there is no such flag). For example, x[["salinityFlag"]] returns a vector of salinity flags if x is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to coastline data: [[<-,coastline-method, as.coastline, coastline-class, coastlineBest, coastlineCut, coastlineWorld, download.coastline, plot,coastline-method, read.coastline.openstreetmap, read.coastline.shapefile, subset,coastline-method, summary,coastline-meth

`[[,ctd-method` *Extract Parts of a CTD Object*

Description

The `[[` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'ctd'
x[[i, j, ...]]
```

Arguments

`x`	A `ctd` object, i.e. one inheriting from `ctd-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

Details of the specialized ctd method

Some uses of [[,ctd-method involve direct retrieval of items within the data slot of the ctd object, while other uses involve calculations based on items in that data slot. For an example, all ctd objects should hold an item named temperature in the data slot, so for example x[["temperature"]] will retrieve that item. By contrast, x[["sigmaTheta"]] is taken to be a request to compute σ_θ, and so it yields a call to swTheta(x) even if the data slot of x might happen to contain an item named theta. This can be confusing at first, but it tends to lead to fewer surprises in everyday work, for otherwise the user would be forced to check the contents of any ctd object under analysis, to determine whether that item will be looked up or computed. Nothing is lost in this scheme, since the data within the object are always accessible with oceGetData.

It should be noted that the accessor is set up to retrieve quantities in conventional units. For example, read.ctd.sbe is used on a .cnv file that stores pressure in psi, it will be stored in the same unit within the ctd object, but x[["pressure"]] will return a value that has been converted to decibars. (Users who need the pressure in PSI can use x@data$pressure.) Similarly, temperature is returned in the ITS-90 scale, with a conversion having been performed with T90fromT68, if the object holds temperature in IPTS-68. Again, temperature on the IPTS-68 scale is returned with x@data$temperature.

This preference for computed over stored quantities is accomplished by first checking for computed quantities, and then falling back to the general [[method if no match is found.

Some quantities are optionally computed. For example, some data files (e.g. the one upon which the section dataset is based) store nitrite along with the sum of nitrite and nitrate, the latter with name `NO2+NO3`. In this case, e.g. x[["nitrate"]] will detect the setup, and subtract nitrite from the sum to yield nitrate.

Below is a list of computed quantities, or at least quantities that are typically not stored in data files. (This is a vague statement because Seabird software permits calculation of many of these and hence storage within .cnv files.)

- CT or Conservative Temperature: Conservative Temperature, computed with gsw_CT_from_t in the gsw package.
- density: seawater density, computed with swRho(x). (Note that it may be better to call that function directly, to gain control of the choice of equation of state, etc.)
- depth: Depth in metres below the surface, computed with swDepth(x).
- N2: Square of Brunt-Vaisala frequency, computed with swN2(x).
- potential temperature: Potential temperature in the UNESCO formulation, computed with swTheta(x). This is a synonym for theta.
- Rrho: Density ratio, computed with swRrho(x).
- SA or Absolute Salinity: Absolute Salinity, computed with gsw_SA_from_SP in the gsw package. The calculation involves location as well as measured water properties. If the object x does not containing information on the location, then 30N and 60W is used for the calculation, and a warning is generated.

- sigmaTheta: A form of potential density anomaly, computed with swSigmaTheta(x).
- sigma0 Equal to sigmaTheta, i.e. potential density anomaly referenced to a pressure of 0dbar, computed with swSigma0(x).
- sigma1: Potential density anomaly referenced to a pressure of 1000dbar, computed with swSigma1(x).
- sigma2: Potential density anomaly referenced to a pressure of 2000dbar, computed with swSigma2(x).
- sigma3: Potential density anomaly referenced to a pressure of 3000dbar, computed with swSigma3(x).
- sigma4: potential density anomaly referenced to a pressure of 4000dbar, computed with swSigma4(x).
- SP: Salinity on the Practical Salinity Scale, which is salinity in the data slot.
- spice: a variable that is in some sense orthogonal to density, calculated with swSpice(x).
- SR: Reference Salinity computed with gsw_SR_from_SP in the gsw package.
- Sstar: Preformed Salinity computed with gsw_SR_from_SP in the gsw package. See SA for a note on longitude and latitude.
- theta: potential temperature in the UNESCO formulation, computed with swTheta(x). This is a synonym for potential temperature.
- z: Vertical coordinate in metres above the surface, computed with swZ(x).

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to ctd data: [[<-,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
data(ctd)
head(ctd[["temperature"]])
```

`[[,echosounder-method` *Extract Parts of an Echosounder Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'echosounder'
x[[i, j, ...]]
```

Arguments

`x`	A `echosounder` object, i.e. one inheriting from `echosounder-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the specialized `echosounder` method

If `i` is the string `"Sv"`, the return value is calculated according to

```
Sv <- 20*log10(a) -
  (x@metadata$sourceLevel+x@metadata$receiverSensitivity+x@metadata$transmitPower) +
  20*log10(r) +
  2*absorption*r -
  x@metadata$correction +
  10*log10(soundSpeed*x@metadata$pulseDuration/1e6*psi/2)
```

If `i` is the string `"TS"`,

```
TS <- 20*log10(a) -
  (x@metadata$sourceLevel+x@metadata$receiverSensitivity+x@metadata$transmitPower) +
  40*log10(r) +
  2*absorption*r +
  x@metadata$correction
```

Otherwise, the generic `[[` is used.

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Other things related to echosounder data: `[[<-,echosounder-method`, `as.echosounder`, `echosounder-class`, `echosounder`, `findBottom`, `plot,echosounder-method`, `read.echosounder`, `subset,echosounder-method`, `summary,echosounder-method`

`[[,g1sst-method` *Extract Something From a G1SST Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'g1sst'
x[[i, j, ...]]
```

Arguments

x	A g1sst object, i.e. one inheriting from g1sst-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of i.

First, a check is made as to whether i names one of the standard oce slots, and returns the slot contents if so. Thus, x[["metadata"]] will retrieve the metadata slot, while x[["data"]] and x[["processingLog"]] return those slots.

Next, if i is a string ending in the "Unit", then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named unit, which is an expression, and an item named scale, which is a string describing the measurement scale. If the string ends in " unit", e.g. x[["temperature unit"]], then just the expression is returned, and if it ends in " scale", then just the scale is returned.

Next, if i is a string ending in "Flag", then the corresponding data-quality flag is returned (or NULL if there is no such flag). For example, x[["salinityFlag"]] returns a vector of salinity flags if x is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to g1sst data: [[<-,g1sst-method

`[[,gps-method` *Extract Something From a GPS Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'gps'
x[[i, j, ...]]
```

Arguments

`x`	A gps object, i.e. one inheriting from `gps-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard oce slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Other things related to gps data: `[[<-,gps-method`, `as.gps`, `gps-class`, `plot,gps-method`, `read.gps`, `summary,gps-method`

`[[,ladp-method` *Extract Something From an ladp Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'ladp'
x[[i, j, ...]]
```

Arguments

`x`	A `ladp` object, i.e. one inheriting from `ladp-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

Author(s)

Dan Kelley

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Other things related to `ladp` data: `as.ladp`, `ladp-class`, `plot,ladp-method`, `summary,ladp-method`

Examples

```
data(ctd)
head(ctd[["temperature"]])
```

`[[,landsat-method` *Extract Something From a landsat Object*

Description

Users are isolated from the details of the two-byte storage system by using the `[[` operator.

Usage

```
## S4 method for signature 'landsat'
x[[i, j, ...]]
```

Arguments

`x`	An landsat object, i.e. one inheriting from `landsat-class`.
`i`	The item to extract.
`j`	Optional additional information on the `i` item (ignored).
`...`	Optional additional information (ignored).

Details

Accessing band data. The data may be accessed with e.g. `landsat[["panchromatic"]]`, for the panchromatic band. If a new "band" is added with `landsatAdd`, it may be referred by name. In all cases, a second argument can be provided, to govern decimation. If this is missing, all the relevant data are returned. If this is present and equal to `TRUE`, then the data will be automatically decimated (subsampled) to give approximately 800 elements in the longest side of the matrix. If this is present and numerical, then its value governs decimation. For example, `landsat[["panchromatic",TRUE]]` will auto-decimate, typically reducing the grid width and height from 16000 to about 800. Similarly, `landsat[["panchromatic",10]]` will reduce width and height to about 1600. On machines with limited RAM (e.g. under about 6GB), decimation is a good idea in almost all processing steps. It also makes sense for plotting, and in fact is done through the `decimate` argument of `plot,landsat-method`.

Accessing derived data. One may retrieve several derived quantities that are calculated from data stored in the object: `landsat[["longitude"]]` and `landsat[["latitude"]]` give pixel locations. Accessing `landsat[["temperature"]]` creates an estimate of ground temperature as follows (see [4]). First, the "count value" in band 10, denoted b_{10} say, is scaled with coefficients stored in the image metadata using $\lambda_L = b_{10}M_L + A_L$ where M_L and A_L are values stored in the metadata (e.g. the first in `landsat@metadata$header$radiance_mult_band_10`) Then the result is used, again with coefficients in the metadata, to compute Celcius temperature $T = K_2/ln(\epsilon K_1/\lambda_L + 1) - 273.15$. The value of the emissivity ϵ is set to unity by `read.landsat`, although it can be changed easily later, by assigning a new value to `landsat@metadata$emissivity`. The default emissivity value set by `read.landsat` is from [11], and is within the oceanic range suggested by [5]. Adjustment is as simple as altering `landsat@metadata$emissivity`. This value can be a single number meant to apply for the whole image, or a matrix with dimensions matching those of band 10. The matrix case is probably more useful for images of land, where one might wish to account for the different emissivities of soil and vegetation, etc.; for example, Table 4 of [9] lists

0.9668 for soil and 0.9863 for vegetation, while Table 5 of [10] lists 0.971 and 0.987 for the same quantities.

Accessing metadata. Anything in the metadata can be accessed by name, e.g. `landsat[["time"]]`. Note that some items are simply copied over from the source data file and are not altered by e.g. decimation. An exception is the lat-lon box, which is altered by `landsatTrim`.

Author(s)

Dan Kelley

See Also

Other things related to `landsat` data: `landsat-class`, `landsatAdd`, `landsatTrim`, `landsat`, `plot,landsat-method`, `read.landsat`, `summary,landsat-method`

`[[,lisst-method` *Extract Something From a LISST Object*

Description

The `[[` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'lisst'
x[[i, j, ...]]
```

Arguments

x	A lisst object, i.e. one inheriting from `lisst-class`.
i	Character string indicating the name of item to extract.
j	Optional additional information on the `i` item.
...	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of i.

First, a check is made as to whether i names one of the standard oce slots, and returns the slot contents if so. Thus, x[["metadata"]] will retrieve the metadata slot, while x[["data"]] and x[["processingLog"]] return those slots.

Next, if i is a string ending in the "Unit", then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named unit, which is an expression, and an item named scale, which is a string describing the measurement scale. If the string ends in " unit", e.g. x[["temperature unit"]], then just the expression is returned, and if it ends in " scale", then just the scale is returned.

Next, if i is a string ending in "Flag", then the corresponding data-quality flag is returned (or NULL if there is no such flag). For example, x[["salinityFlag"]] returns a vector of salinity flags if x is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to lisst data: [[<-,lisst-method, as.lisst, lisst-class, plot,lisst-method, read.lisst, summary,lisst-method

---

[[,lobo-method *Extract Something From a LOBO Object*

Description

The [[method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related replacement method, [[<-, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, NULL is returned.

Usage

```
## S4 method for signature 'lobo'
x[[i, j, ...]]
```

Arguments

x	A lobo object, i.e. one inheriting from lobo-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of i.

First, a check is made as to whether i names one of the standard oce slots, and returns the slot contents if so. Thus, x[["metadata"]] will retrieve the metadata slot, while x[["data"]] and x[["processingLog"]] return those slots.

Next, if i is a string ending in the "Unit", then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named unit, which is an expression, and an item named scale, which is a string describing the measurement scale. If the string ends in " unit", e.g. x[["temperature unit"]], then just the expression is returned, and if it ends in " scale", then just the scale is returned.

Next, if i is a string ending in "Flag", then the corresponding data-quality flag is returned (or NULL if there is no such flag). For example, x[["salinityFlag"]] returns a vector of salinity flags if x is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to lobo data: [[<-,lobo-method, as.lobo, lobo-class, lobo, plot,lobo-method, subset,lobo-method, summary,lobo-method

`[[,met-method` *Extract Something From a met Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'met'
x[[i, j, ...]]
```

Arguments

`x`	A met object, i.e. one inheriting from `met-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard oce slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Other things related to `met` data: `[[<-,met-method`, `as.met`, `download.met`, `met-class`, `met`, `plot,met-method`, `read.met`, `subset,met-method`, `summary,met-method`

`[[,oce-method` *Extract Something From an oce Object*

Description

The named item is sought first in `metadata`, where an exact match to the name is required. If it is not present in the `metadata` slot, then a partial-name match is sought in the `data` slot. Failing both tests, an exact-name match is sought in a field named `dataNamesOriginal` in the object's `metadata` slot, if that field exists. Failing that, `NULL` is returned.

To get information on the specialized variants of this function, type e.g. `?"[[,adv-method"` for information on extracting data from an object of `adv-class`.

Usage

```
## S4 method for signature 'oce'
x[[i, j, ...]]
```

Arguments

`x`	An oce object
`i`	The item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Examples

```
data(ctd)
ctd[["longitude"]] # in metadata
head(ctd[["salinity"]]) # in data
data(section)
summary(section[["station", 1]])
```

[[,odf-method	*Extract Something From an ODF Object*

Description

The `[[` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'odf'
x[[i, j, ...]]
```

Arguments

`x`	an odf object, i.e. one inheriting from `odf-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to odf data: ODF2oce, ODFNames2oceNames, [[<-,odf-method, odf-class, plot,odf-method, read.ctd.odf, read.odf, subset,odf-method, summary,odf-method

---

[[,rsk-method *Extract Something From a Rsk Object*

Description

The [[method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related replacement method, [[<-, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see “Details of the specialized ... method”). Second, if no match is found, a general function is used (see ‘Details of the general method’). If neither method can locates the requested item, NULL is returned.

Usage

```
## S4 method for signature 'rsk'
x[[i, j, ...]]
```

Arguments

x	A rsk object, i.e. one inheriting from rsk-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Other things related to `rsk` data: `[[<-,rsk-method`, `as.rsk`, `plot,rsk-method`, `read.rsk`, `rsk-class`, `rskPatm`, `rskToc`, `rsk`, `subset,rsk-method`, `summary,rsk-method`

`[[,sealevel-method` *Extract Something From a Sealevel Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'sealevel'
x[[i, j, ...]]
```

Arguments

x	A sealevel object, i.e. one inheriting from sealevel-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of i.

First, a check is made as to whether i names one of the standard oce slots, and returns the slot contents if so. Thus, x[["metadata"]] will retrieve the metadata slot, while x[["data"]] and x[["processingLog"]] return those slots.

Next, if i is a string ending in the "Unit", then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named unit, which is an expression, and an item named scale, which is a string describing the measurement scale. If the string ends in " unit", e.g. x[["temperature unit"]], then just the expression is returned, and if it ends in " scale", then just the scale is returned.

Next, if i is a string ending in "Flag", then the corresponding data-quality flag is returned (or NULL if there is no such flag). For example, x[["salinityFlag"]] returns a vector of salinity flags if x is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method, [[<-,adv-method

Other things related to sealevel data: [[<-,sealevel-method, as.sealevel, plot,sealevel-method, read.sealevel, sealevel-class, sealevelTuktoyaktuk, sealevel, subset,sealevel-method, summary,sealevel-method

`[[,section-method` *Extract Something From a Section Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'section'
x[[i, j, ...]]
```

Arguments

`x`	A `section` object, i.e. one inheriting from `section-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the specialized section method

There are several possibilities, depending on the nature of `i`.

- If `i` is the string `"station"`, then the method will return a `list` of `ctd-class` objects holding the station data. If `j` is also given, it specifies a station (or set of stations) to be returned. if `j` contains just a single value, then that station is returned, but otherwise a list is returned. If `j` is an integer, then the stations are specified by index, but if it is character, then stations are specified by the names stored within their metadata. (Missing stations yield `NULL` in the return value.)
- If `i` is `"station ID"`, then the IDs of the stations in the section are returned.
- If `i` is `"dynamic height"`, then an estimate of dynamic height is returned (as calculated with `swDynamicHeight(x)`).
- If `i` is `"distance"`, then the distance along the section is returned, using `geodDist`.
- If `i` is `"depth"`, then a vector containing the depths of the stations is returned.
- If `i` is `"z"`, then a vector containing the z coordinates is returned.
- If `i` is `"theta"` or `"potential temperature"`, then the potential temperatures of all the stations are returned in one vector. Similarly, `"spice"` returns the property known as spice, using `swSpice`.

- If `i` is a string ending with `"Flag"`, then the characters prior to that ending are taken to be the name of a variable contained within the stations in the section. If this flag is available in the first station of the section, then the flag values are looked up for every station.

If none of the conditions listed above holds, the general method is used (see 'Details of the general method').

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

Author(s)

Dan Kelley

See Also

Other things related to `section` data: `[[<-,section-method`, `as.section`, `handleFlags,section-method`, `plot,section-method`, `read.section`, `section-class`, `sectionAddStation`, `sectionGrid`, `sectionSmooth`, `sectionSort`, `section`, `subset,section-method`, `summary,section-method`

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,tidem-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Examples

```
data(section)
length(section[["latitude"]])
length(section[["latitude", "byStation"]])

data(section)
length(section[["latitude"]])
length(section[["latitude", "byStation"]])
```

`[[,tidem-method` *Extract Something From a Tidem Object*

Description

The `[[` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'tidem'
x[[i, j, ...]]
```

Arguments

`x`	A tidem object, i.e. one inheriting from `tidem-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the specialized `tidem` method

A vector of the frequencies of fitted constituents is recovered with e.g. `x[["frequency"]]`. Similarly, amplitude is recovered with e.g. `x[["amplitude"]]` and phase with e.g. `x[["phase"]]`. If any other string is specified, then the underlying accessor `[[,oce-method`) is used.

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard oce slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,topo-method`, `[[,windrose-method`, `[[<-,adv-method`

Other things related to `tidem` data: `[[<-,tidem-method`, `plot,tidem-method`, `predict.tidem`, `summary,tidem-method`, `tidedata`, `tidem-class`, `tidemAstron`, `tidemVuf`, `tidem`

`[[,topo-method` *Extract Something From a Topo Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'topo'
x[[i, j, ...]]
```

Arguments

x	A topo object, i.e. one inheriting from topo-class.
i	Character string indicating the name of item to extract.
j	Optional additional information on the i item.
...	Optional additional information (ignored).

Details of the specialized topo method

There are no special features for topo-class data; the general method is used directly.

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of i.

First, a check is made as to whether i names one of the standard oce slots, and returns the slot contents if so. Thus, x[["metadata"]] will retrieve the metadata slot, while x[["data"]] and x[["processingLog"]] return those slots.

Next, if i is a string ending in the "Unit", then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named unit, which is an expression, and an item named scale, which is a string describing the measurement scale. If the string ends in " unit", e.g. x[["temperature unit"]], then just the expression is returned, and if it ends in " scale", then just the scale is returned.

Next, if i is a string ending in "Flag", then the corresponding data-quality flag is returned (or NULL if there is no such flag). For example, x[["salinityFlag"]] returns a vector of salinity flags if x is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,windrose-method, [[<-,adv-method

Other things related to topo data: [[<-,topo-method, as.topo, download.topo, plot,topo-method, read.topo, subset,topo-method, summary,topo-method, topo-class, topoInterpolate, topoWorld

Examples

```
data(topoWorld)
dim(topoWorld[['z']])
```

`[[,windrose-method` *Extract Something From a Windrose Object*

Description

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 method for signature 'windrose'
x[[i, j, ...]]
```

Arguments

`x`	A windrose object, i.e. one inheriting from `windrose-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).

Details of the specialized windrose method

There are no special features for `windrose-class` data; the general method is used directly.

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard `oce` slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the `metadata` slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the `data` slot is checked to see if it contains an item with the name indicated by `i`. In this case, a partial match will work; this is accomplished by using `pmatch`.

If none of the above-listed conditions holds, then `NULL` is returned.

See Also

Other functions that extract parts of oce objects: `[[,adp-method`, `[[,adv-method`, `[[,amsr-method`, `[[,argo-method`, `[[,bremen-method`, `[[,cm-method`, `[[,coastline-method`, `[[,ctd-method`, `[[,echosounder-method`, `[[,g1sst-method`, `[[,gps-method`, `[[,ladp-method`, `[[,lisst-method`, `[[,lobo-method`, `[[,met-method`, `[[,odf-method`, `[[,rsk-method`, `[[,sealevel-method`, `[[,section-method`, `[[,tidem-method`, `[[,topo-method`, `[[<-,adv-method`

Other things related to `windrose` data: `[[<-,windrose-method`, `as.windrose`, `plot,windrose-method`, `summary,windrose-method`, `windrose-class`

`[[<-,adp-method` *Replace Parts of an ADP Object*

Description

The `[[<-` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'adp'
x[[i, j, ...]] <- value
```

Arguments

`x`	An adp object, i.e. one inheriting from `adp-class`.
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

In addition to the usual insertion of elements by name, note that e.g. pitch gets stored into pitchSlow.

Author(s)

Dan Kelley

See Also

Other functions that replace parts of oce objects: [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to adp data: [[,adp-method, adp-class, adpEnsembleAverage, adp, as.adp, beamName, beamToXyzAdp, beamToXyzAdv, beamToXyz, beamUnspreadAdp, binmapAdp, enuToOtherAdp, enuToOther, plot,adp-method, read.ad2cp, read.adp.nortek, read.adp.rdi, read.adp.sontek.serial, read.adp.sontek, read.adp, read.aquadoppHR, read.aquadoppProfiler, read.aquadopp, subset,adp-method, summary,adp-method, toEnuAdp, toEnu, velocityStatistics, xyzToEnuAdp, xyzToEnu

---

[[<-,adv-method *Replace Parts of an ADV Object*

Description

Replace Parts of an ADV Object

The `[[` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related replacement method, `[[<-`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be extracted, including derived data, units of measurement, and data-quality flags.

The method uses a two-step process to try to find the requested information. First, a class-specific function is used to try to access the requested information (see "Details of the specialized ... method"). Second, if no match is found, a general function is used (see 'Details of the general method'). If neither method can locates the requested item, `NULL` is returned.

Usage

```
## S4 replacement method for signature 'adv'
x[[i, j, ...]] <- value
```

Arguments

`x`	An adv object, i.e. one inheriting from `adv-class`.
`i`	Character string indicating the name of item to extract.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be inserted into `x`.

Details

If the adv object holds slow variables (i.e. if `timeSlow` is in the `data` slot), then assigning to .e.g. `heading` will not actually assign to a variable of that name, but instead assigns to `headingSlow`. To catch misapplication of this rule, an error message will be issued if the assigned value is not of the same length as `timeSlow`.

Details of the general method

If the specialized method produces no matches, the following generalized method is applied. As with the specialized method, the procedure hinges first on the value of `i`.

First, a check is made as to whether `i` names one of the standard oce slots, and returns the slot contents if so. Thus, `x[["metadata"]]` will retrieve the `metadata` slot, while `x[["data"]]` and `x[["processingLog"]]` return those slots.

Next, if `i` is a string ending in the `"Unit"`, then the characters preceding that string are taken to be the name of an item in the data object, and a list containing the unit is returned. This list consists of an item named `unit`, which is an `expression`, and an item named `scale`, which is a string describing the measurement scale. If the string ends in `" unit"`, e.g. `x[["temperature unit"]]`, then just the expression is returned, and if it ends in `" scale"`, then just the scale is returned.

Next, if `i` is a string ending in `"Flag"`, then the corresponding data-quality flag is returned (or `NULL` if there is no such flag). For example, `x[["salinityFlag"]]` returns a vector of salinity flags if `x` is a ctd object.

If none of the preceding conditions are met, a check is done to see if the metadata slot contains an item with the provided name, and that is returned, if so. A direct match is required for this condition.

Finally, the data slot is checked to see if it contains an item with the name indicated by i. In this case, a partial match will work; this is accomplished by using pmatch.

If none of the above-listed conditions holds, then NULL is returned.

Author(s)

Dan Kelley

See Also

Other functions that extract parts of oce objects: [[,adp-method, [[,adv-method, [[,amsr-method, [[,argo-method, [[,bremen-method, [[,cm-method, [[,coastline-method, [[,ctd-method, [[,echosounder-method, [[,g1sst-method, [[,gps-method, [[,ladp-method, [[,lisst-method, [[,lobo-method, [[,met-method, [[,odf-method, [[,rsk-method, [[,sealevel-method, [[,section-method, [[,tidem-method, [[,topo-method, [[,windrose-method

Other things related to adv data: [[,adv-method, adv-class, adv, beamName, beamToXyz, enuToOtherAdv, enuToOther, plot,adv-method, read.adv.nortek, read.adv.sontek.adr, read.adv.sontek.serial, read.adv.sontek.text, read.adv, subset,adv-method, summary,adv-method, toEnuAdv, toEnu, velocityStatistics, xyzToEnuAdv, xyzToEnu

---

[[<-,amsr-method   *Replace Parts of an AMSR Object*

Description

The [[<- method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'amsr'
x[[i, j, ...]] <- value
```

Arguments

x	An amsr object, i.e. inheriting from amsr-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with `[[` method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other things related to `amsr` data: `amsr-class`, `composite,amsr-method`, `download.amsr`, `plot,amsr-method`, `read.amsr`, `subset,amsr-method`, `summary,amsr-method`

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,cm-method`, `[[<-,coastline-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,gps-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,oce-method`, `[[<-,odf-method`, `[[<-,rsk-method`, `[[<-,sealevel-method`, `[[<-,section-method`, `[[<-,tidem-method`, `[[<-,topo-method`, `[[<-,windrose-method`

`[[<-,argo-method` *Replace Parts of an Argo Object*

Description

The `[[<-` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'argo'
x[[i, j, ...]] <- value
```

Arguments

x	An argo object, i.e. inheriting from argo-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to argo data: [[,argo-method, argo-class, argoGrid, argoNames2oceNames, argo, as.argo, handleFlags,argo-method, plot,argo-method, read.argo, subset,argo-method, summary,argo-method

[[<-,bremen-method *Replace Parts of a Bremen Object*

Description

The [[<- method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'bremen'
x[[i, j, ...]] <- value
```

Arguments

x	An bremen object, i.e. inheriting from bremen-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to bremen data: [[,bremen-method, bremen-class, plot,bremen-method, read.bremen, summary,bremen-method

---

[[<-,cm-method *Replace Parts of a CM Object*

Description

The [[<- method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'cm'
x[[i, j, ...]] <- value
```

Arguments

x	An cm object, i.e. inheriting from cm-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,amsr-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,coastline-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,gps-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,oce-method`, `[[<-,odf-method`, `[[<-,rsk-method`, `[[<-,sealevel-method`, `[[<-,section-method`, `[[<-,tidem-method`, `[[<-,topo-method`, `[[<-,windrose-method`

Other things related to `cm` data: `[[,cm-method`, `as.cm`, `cm-class`, `cm`, `plot,cm-method`, `read.cm`, `subset,cm-method`, `summary,cm-method`

`[[<-,coastline-method` *Replace Parts of a Coastline Object*

Description

The `[[<-` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'coastline'
x[[i, j, ...]] <- value
```

Arguments

`x`	An `coastline` object, i.e. inheriting from `coastline-class`
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with `[[` method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other things related to coastline data: `[[,coastline-method`, `as.coastline`, `coastline-class`, `coastlineBest`, `coastlineCut`, `coastlineWorld`, `download.coastline`, `plot,coastline-method`, `read.coastline.openstreetmap`, `read.coastline.shapefile`, `subset,coastline-method`, `summary,coastline-meth`

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,amsr-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,cm-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,gps-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,oce-method`, `[[<-,odf-method`, `[[<-,rsk-method`, `[[<-,sealevel-method`, `[[<-,section-method`, `[[<-,tidem-method`, `[[<-,topo-method`, `[[<-,windrose-method`

`[[<-,ctd-method` *Replace Parts of a CTD Object*

Description

The `[[<-` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'ctd'
x[[i, j, ...]] <- value
```

Arguments

`x`	A `ctd` object, i.e. one inheriting from `ctd-class`.
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to ctd data: [[,ctd-method, as.ctd, cnvName2oceName, ctd-class, ctdDecimate, ctdFindProfiles, ctdRaw, ctdTrim, ctd, handleFlags,ctd-method, oceNames2whpNames, oceUnits2whpUnits, plot,ctd-method, plotProfile, plotScan, plotTS, read.ctd.itp, read.ctd.odf, read.ctd.sbe, read.ctd.woce.other, read.ctd.woce, read.ctd, subset,ctd-method, summary,ctd-method, woceNames2oceNames, woceUnit2oceUnit, write.ctd

Examples

```
data(ctd)
summary(ctd)
# Move the CTD profile a nautical mile north.
ctd[["latitude"]] <- 1/60 + ctd[["latitude"]] # acts in metadata
# Increase the salinity by 0.01.
ctd[["salinity"]] <- 0.01 + ctd[["salinity"]] # acts in data
summary(ctd)
```

[[<-,echosounder-method

Replace Parts of an Echosounder Object

Description

The [[<- method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'echosounder'
x[[i, j, ...]] <- value
```

Arguments

x	An echosounder object, i.e. inheriting from echosounder-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to echosounder data: [[,echosounder-method, as.echosounder, echosounder-class, echosounder, findBottom, plot,echosounder-method, read.echosounder, subset,echosounder-method, summary,echosounder-method

---

`[[<-,g1sst-method` *Replace Parts of a G1SST Object*

Description

The `[[<-` method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'g1sst'
x[[i, j, ...]] <- value
```

Arguments

`x`	An `g1sst` object, i.e. one inheriting from g1sst-class
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with [[method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to g1sst data: [[,g1sst-method

[[<-,gps-method *Replace Parts of a GPS Object*

Description

The [[<- method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'gps'
x[[i, j, ...]] <- value
```

Arguments

x	An gps object, i.e. inheriting from gps-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,amsr-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,cm-method`, `[[<-,coastline-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,oce-method`, `[[<-,odf-method`, `[[<-,rsk-method`, `[[<-,sealevel-method`, `[[<-,section-method`, `[[<-,tidem-method`, `[[<-,topo-method`, `[[<-,windrose-method`

Other things related to gps data: `[[,gps-method`, `as.gps`, `gps-class`, `plot,gps-method`, `read.gps`, `summary,gps-method`

`[[<-,lisst-method` *Replace Parts of a LISST Object*

Description

Replace Parts of a LISST Object

The `[[<-` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'lisst'
x[[i, j, ...]] <- value
```

Arguments

`x`	An `lisst` object, i.e. inheriting from `lisst-class`
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to lisst data: [[,lisst-method, as.lisst, lisst-class, plot,lisst-method, read.lisst, summary,lisst-method

---

[[<-,lobo-method *Replace Parts of a LOBO Object*

Description

The [[<- method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'lobo'
x[[i, j, ...]] <- value
```

Arguments

x	An lobo object, i.e. inheriting from lobo-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to lobo data: [[,lobo-method, as.lobo, lobo-class, lobo, plot,lobo-method, subset,lobo-method, summary,lobo-method

---

[[<-,met-method *Replace Parts of a met Object*

Description

The [[<- method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'met'
x[[i, j, ...]] <- value
```

Arguments

x	An met object, i.e. inheriting from met-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to met data: [[,met-method, as.met, download.met, met-class, met, plot,met-method, read.met, subset,met-method, summary,met-method

`[[<-,oce-method` *Replace Parts of an Oce Object*

Description

The `[[<-` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'oce'
x[[i, j, ...]] <- value
```

Arguments

`x`	An oce object, i.e. inheriting from `oce-class`.
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with `[[` method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,amsr-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,cm-method`, `[[<-,coastline-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,gps-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,odf-method`, `[[<-,rsk-method`, `[[<-,sealevel-method`, `[[<-,section-method`, `[[<-,tidem-method`, `[[<-,topo-method`, `[[<-,windrose-method`

`[[<-,odf-method` *Replace Parts of an ODF Object*

Description

The `[[<-` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'odf'
x[[i, j, ...]] <- value
```

Arguments

`x`	an `odf` object, i.e. inheriting from `odf-class`
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with `[[` method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Other things related to odf data: ODF2oce, ODFNames2oceNames, [[,odf-method, odf-class, plot,odf-method, read.ctd.odf, read.odf, subset,odf-method, summary,odf-method

---

[[<-,rsk-method — *Replace Parts of a Rsk Object*

Description

The [[<- method works for all oce objects, i.e. objects inheriting from oce-class. The purpose, as with the related extraction method, [[, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'rsk'
x[[i, j, ...]] <- value
```

Arguments

x	An rsk object, i.e. inheriting from rsk-class
i	The item to replace.
j	Optional additional information on the i item.
...	Optional additional information (ignored).
value	The value to be placed into x, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,amsr-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,cm-method`, `[[<-,coastline-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,gps-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,oce-method`, `[[<-,odf-method`, `[[<-,sealevel-method`, `[[<-,section-method`, `[[<-,tidem-method`, `[[<-,topo-method`, `[[<-,windrose-method`

Other things related to `rsk` data: `[[,rsk-method`, `as.rsk`, `plot,rsk-method`, `read.rsk`, `rsk-class`, `rskPatm`, `rskToc`, `rsk`, `subset,rsk-method`, `summary,rsk-method`

`[[<-,sealevel-method` *Replace Parts of a Sealevel Object*

Description

The `[[<-` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'sealevel'
x[[i, j, ...]] <- value
```

Arguments

`x`	An `sealevel` object, i.e. inheriting from `sealevel-class`
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with `[[` method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,amsr-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,cm-method`, `[[<-,coastline-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,gps-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,oce-method`, `[[<-,odf-method`, `[[<-,rsk-method`, `[[<-,section-method`, `[[<-,tidem-method`, `[[<-,topo-method`, `[[<-,windrose-method`

Other things related to `sealevel` data: `[[,sealevel-method`, `as.sealevel`, `plot,sealevel-method`, `read.sealevel`, `sealevel-class`, `sealevelTuktoyaktuk`, `sealevel`, `subset,sealevel-method`, `summary,sealevel-method`

`[[<-,section-method` *Replace Parts of a Section Object*

Description

The `[[<-` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'section'
x[[i, j, ...]] <- value
```

Arguments

`x`	A `section` object, i.e. inheriting from `section-class`
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with [[method, the procedure works in steps.

First, the metadata slot of x is checked to see whether it contains something named with i. If so, then the named item is replaced with value.

Otherwise, if the string value of i ends in Unit, then the characters preceding that are taken as the name of a variable, and the metadata slot of x is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

Author(s)

Dan Kelley

See Also

Other things related to section data: [[,section-method, as.section, handleFlags,section-method, plot,section-method, read.section, section-class, sectionAddStation, sectionGrid, sectionSmooth, sectionSort, section, subset,section-method, summary,section-method

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,tidem-method, [[<-,topo-method, [[<-,windrose-method

Examples

```
# 1. Change section ID from a03 to A03
data(section)
section[["sectionId"]]
section[["sectionId"]] <- toupper(section[["sectionId"]])
section[["sectionId"]]
# 2. Add a millidegree to temperatures at station 10
section[["station", 10]][["temperature"]] <-
    1e-3 + section[["station", 10]][["temperature"]]
```

`[[<-,tidem-method` *Replace Parts of a Tidem Object*

Description

The `[[<-` method works for all `oce` objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of `oce` objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'tidem'
x[[i, j, ...]] <- value
```

Arguments

`x`	An `tidem` object, i.e. inheriting from `tidem-class`
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with `[[` method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,amsr-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,cm-method`, `[[<-,coastline-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,gps-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,oce-method`, `[[<-,odf-method`, `[[<-,rsk-method`, `[[<-,sealevel-method`, `[[<-,section-method`, `[[<-,topo-method`, `[[<-,windrose-method`

Other things related to `tidem` data: `[[,tidem-method`, `plot,tidem-method`, `predict.tidem`, `summary,tidem-method`, `tidedata`, `tidem-class`, `tidemAstron`, `tidemVuf`, `tidem`

`[[<-,topo-method` *Replace Parts of a Topo Object*

Description

The `[[<-` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'topo'
x[[i, j, ...]] <- value
```

Arguments

`x`	An `topo` object, i.e. inheriting from `topo-class`
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with `[[` method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if `i` ends in `Flag`, then quality-control flags are set up as defined by `result`, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the `data` slot of `x`. (This is done with `pmatch`.) The first item found (if any) is then updated to hold the value `result`.

If none of these conditions is met, a warning is issued.

See Also

Other things related to topo data: `[[,topo-method`, `as.topo`, `download.topo`, `plot,topo-method`, `read.topo`, `subset,topo-method`, `summary,topo-method`, `topo-class`, `topoInterpolate`, `topoWorld`

Other functions that replace parts of oce objects: `[[<-,adp-method`, `[[<-,amsr-method`, `[[<-,argo-method`, `[[<-,bremen-method`, `[[<-,cm-method`, `[[<-,coastline-method`, `[[<-,ctd-method`, `[[<-,echosounder-method`, `[[<-,g1sst-method`, `[[<-,gps-method`, `[[<-,lisst-method`, `[[<-,lobo-method`, `[[<-,met-method`, `[[<-,oce-method`, `[[<-,odf-method`, `[[<-,rsk-method`, `[[<-,sealevel-method`, `[[<-,section-method`, `[[<-,tidem-method`, `[[<-,windrose-method`

`[[<-,windrose-method` *Replace Parts of a Windrose Object*

Description

The `[[<-` method works for all oce objects, i.e. objects inheriting from `oce-class`. The purpose, as with the related extraction method, `[[`, is to insulate users from the internal details of oce objects, by looking for items within the various storage slots of the object. Items not actually stored can also be replaced, including units and data-quality flags.

Usage

```
## S4 replacement method for signature 'windrose'
x[[i, j, ...]] <- value
```

Arguments

`x`	An `windrose` object, i.e. inheriting from `windrose-class`
`i`	The item to replace.
`j`	Optional additional information on the `i` item.
`...`	Optional additional information (ignored).
`value`	The value to be placed into `x`, somewhere.

Details

As with `[[` method, the procedure works in steps.

First, the `metadata` slot of `x` is checked to see whether it contains something named with `i`. If so, then the named item is replaced with `value`.

Otherwise, if the string value of `i` ends in `Unit`, then the characters preceding that are taken as the name of a variable, and the `metadata` slot of `x` is updated to store that unit, e.g.

```
x[["temperatureUnits"]] <- list(unit=expression(degree*F),scale="")
```

Similarly, if i ends in Flag, then quality-control flags are set up as defined by result, e.g.

```
x[["temperatureFlags"]] <- c(2,4,2,2)
```

Otherwise, a partial string match is sought among the names of items in the data slot of x. (This is done with pmatch.) The first item found (if any) is then updated to hold the value result.

If none of these conditions is met, a warning is issued.

See Also

Other functions that replace parts of oce objects: [[<-,adp-method, [[<-,amsr-method, [[<-,argo-method, [[<-,bremen-method, [[<-,cm-method, [[<-,coastline-method, [[<-,ctd-method, [[<-,echosounder-method, [[<-,g1sst-method, [[<-,gps-method, [[<-,lisst-method, [[<-,lobo-method, [[<-,met-method, [[<-,oce-method, [[<-,odf-method, [[<-,rsk-method, [[<-,sealevel-method, [[<-,section-method, [[<-,tidem-method, [[<-,topo-method

Other things related to windrose data: [[,windrose-method, as.windrose, plot,windrose-method, summary,windrose-method, windrose-class

Index