Package 'xts'

January 21, 2024

Type Package

Title eXtensible Time Series

Version 0.13.2

Depends R (>= 3.6.0), zoo (>= 1.7-12)

Imports methods

LinkingTo zoo

Suggests timeSeries, timeDate, tseries, chron, tinytest

LazyLoad yes

Description Provide for uniform handling of R's different time-based data classes by extending zoo, maximizing native format information preservation and allowing for user level customization and extension, while simplifying cross-class interoperability.

License GPL (>= 2)

URL https://joshuaulrich.github.io/xts/, https://github.com/joshuaulrich/xts

BugReports https://github.com/joshuaulrich/xts/issues

NeedsCompilation yes

Author Jeffrey A. Ryan [aut, cph],
Joshua M. Ulrich [cre, aut],
Ross Bennett [ctb],
Corwin Joy [ctb]

Maintainer Joshua M. Ulrich <josh.m.ulrich@gmail.com>

Repository CRAN

Date/Publication 2024-01-21 16:10:02 UTC

R topics documented:

xts-package *xts: extensible time-series*

Description

Extensible time series class and methods, extending and behaving like zoo.

Details

Easily convert one of R's many time-series (and non-time-series) classes to a true time-based object which inherits all of zoo's methods, while allowing for new time-based tools where appropriate.

Additionally, one may use **xts** to create new objects which can contain arbitrary attributes named during creation as name=value pairs.

Author(s)

Jeffrey A. Ryan and Joshua M. Ulrich

Maintainer: Joshua M. Ulrich <josh.m.ulrich@gmail.com>

See Also

`xts` `as.xts` `reclass` `zoo`

.parseISO8601 *Internal ISO 8601:2004(e) Time Parser*

Description

This function is used internally in the subsetting mechanism of xts. The function is unexported, though documented for use with xts subsetting.

Usage

```
.parseISO8601(x, start, end, tz="")

.makeISO8601(x)
```

Arguments

`x`	For .parseISO8601(x), a character string conforming to the ISO 8601:2004(e) rules. For .makeISO8601(x), `x` should be a time-like object with `start` and `end` methods.
`start`	lower constraint on range
`end`	upper constraint of range
`tz`	timezone (tzone) to use internally

Details

This function replicates most of the ISO standard for expressing time and time-based ranges in a universally accepted way.

The best documentation is now the official ISO page as well as the Wikipedia entry for ISO 8601:2004.

The basic idea is to create the endpoints of a range, given a string representation. These endpoints are aligned in POSIXct time to the zero second of the day at the beginning, and the 59.9999th second of the 59th minute of the 23rd hour of the final day.

For dates prior to the epoch (1970-01-01) the ending time is aligned to the 59.0000 second. This is due to a bug/feature in the R implementation of asPOSIXct and mktime0 at the C-source level. This limits the precision of ranges prior to 1970 to 1 minute granularity with the current **xts** workaround.

Recurring times over multiple days may be specified using the T notation. See the examples for details.

Value

A list of length two, with an entry named 'first.time' and one names 'last.time'.

For .makeISO8601, a character vector of length one describing the ISO-style format for a given time-based object.

Note

There is no checking done to test for a properly constructed ISO format string. This must be correctly entered by the user, lest bad things may happen.

When using durations, it is important to note that the time of the duration specified is not necessarily the same as the realized periods that may be returned when applied to an irregular time series. This is not a bug, rather it is a standards and implementation gotcha.

Author(s)

Jeffrey A. Ryan

References

https://en.wikipedia.org/wiki/ISO_8601
https://www.iso.org/iso-8601-date-and-time-format.html

Examples

```
# the start and end of 2000
.parseISO8601('2000')

# the start of 2000 and end of 2001
.parseISO8601('2000/2001')

# May 1, 2000 to Dec 31, 2001
.parseISO8601('2000-05/2001')
```

```
# May 1, 2000 to end of Feb 2001
.parseISO8601('2000-05/2001-02')

# Jan 1, 2000 to Feb 29, 2000; note the truncated time on the LHS
.parseISO8601('2000-01/02')

# 8:30 to 15:00 (used in xts subsetting to extract recurring times)
.parseISO8601('T08:30/T15:00')
```

`addEventLines` *Add vertical lines to an existing xts plot*

Description

Add vertical lines and labels to an existing xts plot

Usage

```
addEventLines(events, main = "", on = 0, lty = 1, lwd = 1, col = 1, ...)
```

Arguments

`events`	xts object of events and their associated labels. It is assumed that the first column of `events` is the event description/label.
`main`	main title for a new panel if drawn.
`on`	panel number to draw on. A new panel will be drawn if `on=NA`. The default, `on=0`, will add to the active panel. The active panel is defined as the panel on which the most recent action was performed. Note that only the first element of `on` is checked for the default behavior to add to the last active panel.
`lty`	set the line type, same as in `par`.
`lwd`	set the line width, same as in `par`.
`col`	color palette to use, set by default to rational choices.
`...`	any other passthrough parameters to `text` to control how the event labels are drawn

Author(s)

Ross Bennett

Examples

```
## Not run:
library(xts)
data(sample_matrix)
sample.xts <- as.xts(sample_matrix)
events <- xts(letters[1:3],
              as.Date(c("2007-01-12", "2007-04-22", "2007-06-13")))
```

```
plot(sample.xts[,4])
addEventLines(events, srt=90, pos=2)

## End(Not run)
```

---

addLegend *Add Legend*

Description

Add Legend

Usage

```
addLegend(legend.loc = "topright", legend.names = NULL, col = NULL,
  ncol = 1, on = 0, ...)
```

Arguments

legend.loc	legend.loc places a legend into one of nine locations on the chart: bottomright, bottom, bottomleft, left, topleft, top, topright, right, or center.
legend.names	character vector of names for the legend. If NULL, the column names of the current plot object are used.
col	fill colors for the legend. If NULL, the colorset of the current plot object data is used.
ncol	number of columns for the legend
on	panel number to draw on. A new panel will be drawn if on=NA. The default, on=0, will add to the active panel. The active panel is defined as the panel on which the most recent action was performed. Note that only the first element of on is checked for the default behavior to add to the last active panel.
...	any other passthrough parameters to legend.

Author(s)

Ross Bennett

addPanel *Add a panel to an existing xts plot*

Description

Apply a function to the data of an existing xts plot object and plot the result. FUN should have arguments x or R for the data of the existing xts plot object to be passed to. All other additional arguments for FUN are passed through ....

Usage

```
addPanel(FUN, main = "", on = NA, type = "l", col = NULL, lty = 1,
  lwd = 1, pch = 1, ...)
```

Arguments

FUN	an xts object to plot.
main	main title for a new panel if drawn.
on	panel number to draw on. A new panel will be drawn if on=NA.
type	the type of plot to be drawn, same as in plot.
col	color palette to use, set by default to rational choices.
lty	set the line type, same as in par.
lwd	set the line width, same as in par.
pch	the type of plot to be drawn, same as in par.
...	additional named arguments passed through to FUN and any other graphical passthrough parameters.

Author(s)

Ross Bennett

Examples

```
library(xts)
data(sample_matrix)
sample.xts <- as.xts(sample_matrix)

calcReturns <- function(price, method = c("discrete", "log")){
  px <- try.xts(price)
  method <- match.arg(method)[1L]
  returns <- switch(method,
    simple = ,
    discrete = px / lag(px) - 1,
    compound = ,
    log = diff(log(px)))
  reclass(returns, px)
```

```
}

# plot the Close
plot(sample.xts[,"Close"])
# calculate returns
addPanel(calcReturns, method="discrete", type="h")
# Add simple moving average to panel 1
addPanel(rollmean, k=20, on=1)
addPanel(rollmean, k=40, col="blue", on=1)
```

---

addPolygon *Add a polygon to an existing xts plot*

Description

Draw a polygon on an existing xts plot by specifying a time series of y coordinates. The xts index is used for the x coordinates and the first two columns are the upper and lower y coordinates, respectively.

Usage

```
addPolygon(x, y = NULL, main = "", on = NA, col = NULL, ...)
```

Arguments

x	an xts object to plot. Must contain 2 columns for the upper and lower y coordinates for the polygon. The first column is interpreted as the upper y coordinates and the second column as the lower y coordinates.
y	NULL, not used
main	main title for a new panel if drawn.
on	panel number to draw on. A new panel will be drawn if on=NA.
col	color palette to use, set by default to rational choices.
...	passthru parameters to par

Author(s)

Ross Bennett

References

Based on code by Dirk Eddelbuettel from http://dirk.eddelbuettel.com/blog/2011/01/16/

Examples

```
## Not run:
library(xts)
data(sample_matrix)
x <- as.xts(sample_matrix)[,1]
ix <- index(x["2007-02"])
shade <- xts(matrix(rep(range(x), each = length(ix)), ncol = 2), ix)

plot(x)

# set on = -1 to draw the shaded region *behind* the main series
addPolygon(shade, on = -1, col = "lightgrey")

## End(Not run)
```

`addSeries` *Add a time series to an existing xts plot*

Description

Add a time series to an existing xts plot

Usage

```
addSeries(x, main = "", on = NA, type = "l", col = NULL, lty = 1,
  lwd = 1, pch = 1, ...)
```

Arguments

`x`	an xts object to plot.
`main`	main title for a new panel if drawn.
`on`	panel number to draw on. A new panel will be drawn if `on=NA`.
`type`	the type of plot to be drawn, same as in `plot`.
`col`	color palette to use, set by default to rational choices.
`lty`	set the line type, same as in `par`.
`lwd`	set the line width, same as in `par`.
`pch`	the type of plot to be drawn, same as in `par`.
`...`	any other passthrough graphical parameters.

Author(s)

Ross Bennett

`align.time` *Align seconds, minutes, and hours to beginning of next period.*

Description

Change timestamps to the start of the next period, specified in multiples of seconds.

Usage

```
align.time(x, ...)

## S3 method for class 'xts'
align.time(x, n=60, ...)

shift.time(x, n=60, ...)

adj.time(x, ...)
```

Arguments

`x`	object to align
`n`	number of seconds to adjust by
`...`	additional arguments. See details.

Details

This function is an S3 generic. The result is to round up to the next period determined by `n` `modulo` `x`.

Value

A new object of class(x)

Author(s)

Jeffrey A. Ryan with input from Brian Peterson

See Also

`to.period`

Examples

```
x <- Sys.time() + 1:1000

# every 10 seconds
align.time(x, 10)

# align to next whole minute
align.time(x, 60)

# align to next whole 10 min interval
align.time(x, 10 * 60)
```

`apply.monthly` *Apply Function over Calendar Periods*

Description

Apply a specified function to each distinct period in a given time series object.

Usage

```
apply.daily(x, FUN, ...)
apply.weekly(x, FUN, ...)
apply.monthly(x, FUN, ...)
apply.quarterly(x, FUN, ...)
apply.yearly(x, FUN, ...)
```

Arguments

`x`	an time-series object coercible to xts
`FUN`	an R function
`...`	additional arguments to FUN

Details

Simple mechanism to apply a function to non-overlapping time periods, e.g. weekly, monthly, etc. Different from rolling functions in that this will subset the data based on the specified time period (implicit in the call), and return a vector of values for each period in the original data.

Essentially a wrapper to the **xts** functions `endpoints` and `period.apply`, mainly as a convenience.

Value

A vector of results produced by `FUN`, corresponding to the appropriate periods.

Note

When FUN = mean the results will contain one column for every column in the input, which is different from other math functions (e.g. median, sum, prod, sd, etc.).

FUN = mean works by column because the default method stats::mean used to work by column for matrices and data.frames. R Core changed the behavior of mean to always return one column in order to be consistent with the other math functions. This broke some **xts** dependencies and mean.xts was created to maintain the original behavior.

Using FUN = mean will print a message that describes this inconsistency. To avoid the message and confusion, use FUN = colMeans to calculate means by column and use FUN = function(x) mean to calculate one mean for all the data. Set options(xts.message.period.apply.mean = FALSE) to suppress this message.

Author(s)

Jeffrey A. Ryan

See Also

endpoints, period.apply, to.monthly

Examples

```
xts.ts <- xts(rnorm(231),as.Date(13514:13744,origin="1970-01-01"))

start(xts.ts)
end(xts.ts)

apply.monthly(xts.ts,colMeans)
apply.monthly(xts.ts,function(x) var(x))
```

---

as.environment.xts *Coerce an 'xts' Object to an Environment by Column*

Description

Method to automatically convert an 'xts' object to an environment containing vectors representing each column of the original xts object. Each objects will be named according to the column name it is exracted by.

Usage

```
## S3 method for class 'xts'
as.environment(x)
```

Arguments

x	an xts object

Details

An experimental tool to convert xts objects into environments for simplifying use withing the standard R formula/data paradigm.

Value

An environment containing ncol(x) vectors extracted by column from x. Note that environments do not preserve (or have knowledge) of column position, a.k.a order.

Author(s)

Jeffrey A. Ryan

Examples

```
x <- xts(1:10, Sys.Date()+1:10)
colnames(x) <- "X"
y <- xts(1:10, Sys.Date()+1:10)
colnames(x) <- "Y"
xy <- cbind(x,y)
colnames(xy)
e <- as.environment(xy)    # currently using xts-style positive k
ls(xy)
ls.str(xy)
```

---

as.xts *Convert Object To And From Class xts*

Description

Conversion functions to coerce data objects of arbitrary classes to class xts and back, without losing any attributes of the original format.

Usage

```
as.xts(x, ...)
xtsible(x)

Reclass(x)

try.xts(x, ..., error = TRUE)
reclass(x, match.to, error = FALSE, ...)
```

Arguments

x	data object to convert. See details for supported types
match.to	xts object whose attributes will be passed to x
error	error handling option. See Details.
...	additional parameters or attributes

Details

A simple and reliable way to convert many different objects into a uniform format for use within R.

It is possible with a call to `as.xts` to convert objects of class `timeSeries`, `ts`, `irts`, `matrix`, `data.frame`, and `zoo`.

`xtsible` safely checks whether an object can be converted to an `xts` object; returning TRUE on success and FALSE otherwise.

The help file `as.xts.methods` lists all available xts methods and arguments specific to each coercible type.

Additional name=value pairs may be passed to the function to be added to the new object. A special print.xts method will assure that the attributes are hidden from view, but will be available via R's standard `attr` function, as well as the `xtsAttributes` function.

The returned object will preserve all relevant attribute/slot data within itself, allowing for temporary conversion to use zoo and xts compatible methods. A call to `reclass` returns the object to its original class, with all original attributes intact - unless otherwise changed.

It should be obvious, but any attributes added via the ... argument will not be carried back to the original data object, as there would be no available storage slot/attribute.

`Reclass` is designed for top-level use, where it is desirable to have the object returned from an arbitrary function in the same class as the object passed in. Most functions within R are not designed to return objects matching the original object's class. While this tool is highly experimental at present, it attempts to handle conversion and reconversion transparently. The caveats are that the original object must be coercible to `xts`, the returned object must be of the same row length as the original object, and that the object to reconvert to is the first argument to the function being wrapped.

`try.xts` and `reclass` are functions that enable external developers access to the reclassing tools within **xts** to help speed development of time-aware functions, as well as provide a more robust and seemless end-user experience, regardless of the end-user's choice of data-classes.

The `error` argument to try.xts accepts a logical value, indicating where an error should be thrown, a character string allowing for custom error messages to be displayed, or a function of the form `f(x, ...)`, to be called upon construction error.

See the accompanying vignette for more details on the above usage and the package in general.

Value

An S3 object of class `xts`.

In the case of `Reclass` and `reclass`, the object returned will be of the original class as identified by `CLASS`.

Author(s)

Jeffrey A. Ryan

See Also

`xts`,`as.xts.methods`

as.xts.methods *Convert Object To And From Class xts*

Description

Conversion S3 methods to coerce data objects of arbitrary classes to class `xts` and back, without losing any attributes of the original format.

Usage

```
## S3 method for class 'xts'
as.xts(x,...,.RECLASS=FALSE)


## S3 method for class 'timeSeries'
as.xts(x, dateFormat="POSIXct", FinCenter, recordIDs,
    title, documentation, ..., .RECLASS=FALSE)

## S3 method for class 'zoo'
as.xts(x, order.by=index(x), frequency=NULL, ..., .RECLASS=FALSE)

## S3 method for class 'ts'
as.xts(x, dateFormat,...,.RECLASS=FALSE)

## S3 method for class 'data.frame'
as.xts(x, order.by, dateFormat="POSIXct",
    frequency=NULL, ...,.RECLASS=FALSE)

## S3 method for class 'matrix'
as.xts(x, order.by, dateFormat="POSIXct",
    frequency=NULL, ..., .RECLASS=FALSE)
```

Arguments

`x`	data object to convert. See details for supported types
`dateFormat`	what format should the dates be converted to
`FinCenter`	see timeSeries help
`recordIDs`	see timeSeries help
`title`	see timeSeries help
`documentation`	see timeSeries help
`order.by`	see zoo help
`frequency`	see zoo help
`...`	additional parameters or attributes
`.RECLASS`	should conversion be reversible?

Details

A simple and reliable way to convert many different objects into a uniform format for use within R.

It is possible with a call to `as.xts` to convert objects of class `timeSeries`, `ts`, `matrix`, `data.frame`, and `zoo`.

Additional name=value pairs may be passed to the function to be added to the new object. A special print.xts method will assure that the attributes are hidden from view, but will be available via R's standard `attr` function.

If `.RECLASS=TRUE`, the returned object will preserve all relevant attribute/slot data within itself, allowing for temporary conversion to use zoo and xts compatible methods. A call to `reclass` returns the object to its original class, with all original attributes intact - unless otherwise changed. This is the default behavior when `try.xts` is used for conversion, and should not be altered by the user; i.e. don't touch it unless you are aware of the consequences.

It should be obvious, but any attributes added via the ... argument will not be carried back to the original data object, as there would be no available storage slot/attribute.

Value

An S3 object of class `xts`.

Author(s)

Jeffrey A. Ryan

See Also

`xts`, `zoo`

Examples

```
## Not run:
# timeSeries
library(timeSeries)
x <- timeSeries(1:10, 1:10)

str( as.xts(x) )
str( reclass(as.xts(x)) )
str( try.xts(x) )
str( reclass(try.xts(x)) )

## End(Not run)
```

axTicksByTime *Compute x-Axis Tickmark Locations by Time*

Description

Compute x-axis tickmarks like `axTicks` in base but with respect to time. Additionally the first argument is the object indexed by time which you are looking to derive tickmark locations for.

It is possible to specify the detail you are seeking, or by passing 'auto' to the `ticks.on` argument, to get a best heuristic fit.

Usage

```
axTicksByTime(x, ticks.on='auto', k = 1,
              labels=TRUE, format.labels=TRUE, ends=TRUE,
              gt = 2, lt = 30)
```

Arguments

`x`	the object indexed by time, or a vector of times/dates
`ticks.on`	what to break on
`k`	frequency of breaks
`labels`	should a labeled vector be returned
`format.labels`	format labels - may be format to use
`ends`	should the ends be adjusted
`gt`	lower bound on number of breaks
`lt`	upper bound on number of breaks

Details

This function is written for internal use, and documented for those wishing to use outside of the internal function uses. In general it is most unlikely that the end user will call this function directly.

The `format.labels` argument allows for standard formatting like that used in `format`, `strptime`, and `strftime`.

Value

A numeric vector of index element locations where tick marks should be drawn. These are *locations* (e.g. 1, 2, 3, ...), *not* the index timestamps.

If possible, the result will be named using formatted values from the index timestamps. The names will be used for the tick mark labels.

Author(s)

Jeffrey A. Ryan

See Also

endpoints

Examples

```
data(sample_matrix)
axTicksByTime(as.xts(sample_matrix),'auto')
axTicksByTime(as.xts(sample_matrix),'weeks')
axTicksByTime(as.xts(sample_matrix),'months',7)
```

CLASS *Extract and Set .CLASS Attribute*

Description

Simple extraction and replacement function to access xts .CLASS attribute. The .CLASS attribute is used by reclass to transform an xts object back to its original class.

Usage

```
CLASS(x)

CLASS(x) <- value
```

Arguments

x	an xts object
value	the new .CLASS value to assign

Details

It is not recommended that CLASS be called in daily use. While it may be possible to coerce objects to other classes than originally derived from, there is little, if any, chance that the reclass function will perform as expected.

It is best to use the traditional as methods.

Value

Called for its side-effect of changing the .CLASS attribute

Author(s)

Jeffrey A. Ryan

See Also

as.xts,reclass

coredata.xts *Extract/Replace Core Data of an xts Object*

Description

Mechanism to extract and replace the core data of an xts object.

Usage

```
## S3 method for class 'xts'
coredata(x, fmt=FALSE, ...)

xcoredata(x,...)
xcoredata(x) <- value
```

Arguments

x	an xts object
fmt	should the rownames be formated in a non-standard way
value	non-core attributes to assign
...	further arguments [unused]

Details

Extract coredata of an xts object - removing all attributes except dim and dimnames and returning a matrix object with rownames converted from the index of the xts object.

The fmt argument, if TRUE, allows the internal index formatting specified by the user to be used. Alternatively, it may be a valid formatting string to be passed to format. Setting to FALSE will return the row names by simply coercing the index class to a character string in the default manner.

xcoredata is the functional complement to coredata, returning all of the attributes normally removed by coredata. Its purpose, along with the replacement function xcoredata<- is primarily for use by developers using **xts** to allow for internal replacement of values removed during use of non xts-aware functions.

Value

Returns either a matrix object for coredata, or a list of named attributes.

The replacement functions are called for their side-effects.

Author(s)

Jeffrey A. Ryan

See Also

coredata, xtsAttributes

Examples

```
data(sample_matrix)
x <- as.xts(sample_matrix, myattr=100)
coredata(x)
xcoredata(x)
```

`diff.xts` *Lags and Differences of xts Objects*

Description

Methods for computing lags and differences on `xts` objects. This matches most of the functionality of **zoo** methods, with some default argument changes.

Usage

```
## S3 method for class 'xts'
lag(x, k = 1, na.pad = TRUE, ...)



## S3 method for class 'xts'
diff(x, lag = 1, differences = 1, arithmetic = TRUE, log = FALSE, na.pad = TRUE, ...)
```

Arguments

`x`	an `xts` object
`k`	period to lag over
`lag`	period to difference over
`differences`	order of differencing
`arithmetic`	should arithmetic or geometric differencing be used
`log`	should (geometric) log differences be returned
`na.pad`	pad vector back to original size
`...`	additional arguments

Details

The primary motivation for having methods specific to `xts` was to make use of faster C-level code within xts. Additionally, it was decided that `lag`'s default behavior should match the common time-series interpretation of that operator — specifically that a value at time 't' should be the value at time 't-1' for a positive lag. This is different than `lag.zoo` as well as `lag.ts`.

Another notable difference is that `na.pad` is set to TRUE by default, to better reflect the transformation visually and within functions requiring positional matching of data.

Backwards compatability with zoo can be achieved by setting `options(xts.compat.zoo.lag=TRUE)`. This will change the defaults of lag.xts to k=-1 and na.pad=FALSE.

Value

An xts object reflected the desired lag and/or differencing.

Author(s)

Jeffrey A. Ryan

References

https://en.wikipedia.org/wiki/Lag

Examples

```
x <- xts(1:10, Sys.Date()+1:10)
lag(x)    # currently using xts-style positive k


lag(x, k=2)


lag(x, k=-1, na.pad=FALSE) # matches lag.zoo(x, k=1)

diff(x)
diff(x, lag=1)
diff(x, diff=2)
diff(diff(x))
```

---

dimnames.xts *Dimnames of an xts Object*

Description

Get or set dimnames of an xts object.

Usage

```
## S3 method for class 'xts'
dimnames(x)

## S3 replacement method for class 'xts'
dimnames(x) <- value
```

Arguments

x	an xts object
value	a list object of length two. See Details.

Details

The functions `dimnames.xts` and `dimnames<-.xts` are methods for the base functions `dimnames` and `dimnames<-`.

`xts` objects by design are intended for lightweight management of time-indexed data.

Rownames are redundant in this design, as well as quite burdensome with respect to memory consumption and internal copying costs.

`rownames` and `colnames` in R make use of `dimnames` method dispatch internally, and thus require only modifications to dimnames to enforce the `xts` no rownames requirement.

To prevent accidental setting of rownames, `dimnames<-` for `xts` will simply set the rownames to `NULL` when invoked, regardless of attempts to set otherwise.

This is done for internal compatibility reasons, as well as to provide consistency in performance regardless of object use.

User level interaction with either dimnames or rownames will produce a character vector of the index, formatted based on the current specification of `indexFormat`. This occurs within the call by converting the results of calling `index(x)` to a character string, which itself first creates the object type specified internally from the underlying numeric time representation.

Value

A list or character string containing coerced row names and/or actual column names.

Attempts to set rownames on xts objects via rownames or dimnames will silently fail. This is your warning.

Note

All `xts` objects have dimension. There are no `xts` objects representable as named or unnamed vectors.

Author(s)

Jeffrey A. Ryan

See Also

`xts`

Examples

```
x <- xts(1:10, Sys.Date()+1:10)
dimnames(x)
rownames(x)
rownames(x) <- 1:10
rownames(x)
str(x)
```

endpoints *Locate Endpoints by Time*

Description

Extract index locations for an xts object that correspond to the *last* observation in each period specified by on.

Usage

```
endpoints(x, on="months", k=1)
```

Arguments

x	an xts object
on	the periods endpoints to find as a character string
k	along every k-th element - see notes

Details

endpoints returns a numeric vector corresponding to the *last* observation in each period. The vector always begins with zero and ends with the last observation in x.

Periods are always based on the distance from the UNIX epoch (midnight 1970-01-01 UTC), *not the first observation in* x. The examples illustrate this behavior.

Valid values for the argument on include: "us" (microseconds), "microseconds", "ms" (milliseconds), "milliseconds", "secs" (seconds), "seconds", "mins" (minutes), "minutes", "hours", "days", "weeks", "months", "quarters", and "years".

Value

A numeric vector of beginning with 0 and ending with the value equal to the number of observations in the x argument.

Author(s)

Jeffrey A. Ryan

Examples

```
data(sample_matrix)

endpoints(sample_matrix)
endpoints(sample_matrix, "weeks")

### example of how periods are based on the UNIX epoch,
### *not* the first observation of the data series
x <- xts(1:38, yearmon(seq(2018 - 1/12, 2021, 1/12)))
```

```
# endpoints for the end of every other year
ep <- endpoints(x, "years", k = 2)
# Dec-2017 is the end of the *first* year in the data. But when you start from
# Jan-1970 and use every second year end as your endpoints, the endpoints are
# always December of every odd year.
x[ep, ]
```

`first` *Return First or Last n Elements of A Data Object*

Description

A generic function to return the first or last elements or rows of a vector or two-dimensional data object.

A more advanced subsetting is available for zoo objects with indexes inheriting from POSIXt or Date classes.

Usage

```
first(x,...)
last(x,...)

## Default S3 method:
first(x,n=1,keep=FALSE,...)

## Default S3 method:
last(x,n=1,keep=FALSE,...)

## S3 method for class 'xts'
first(x,n=1,keep=FALSE,...)

## S3 method for class 'xts'
last(x,n=1,keep=FALSE,...)
```

Arguments

`x`	1 or 2 dimensional data object
`n`	number of periods to return
`keep`	should removed values be kept?
`...`	additional args - unused

Details

Provides the ability to identify the first or last n rows or observations of a data set. The generic method behaves much like head and tail from **base**, except by default only the *first* or *last* observation will be returned.

The more useful method for the xts class allows for time based subsetting, given an xtsible object.

n may be either a numeric value, indicating the number of observations to return - forward from first, or backwards from last, or it may be a character string describing the number and type of periods to return.

n may be positive or negative, in either numeric or character contexts. When positive it will return the result expected - e.g. last(X,'1 month') will return the last month's data. If negative, all data will be returned *except* for the last month. It is important to note that this is not the same as calling first(X,'1 month') or first(X,'-1 month'). All 4 variations return different subsets of data and have distinct purposes.

If n is a character string, it must be of the form 'n period.type' or 'period.type', where n is a numeric value (defaults to 1 if not provided) describing the number of period.types to move forward (first) or back (last).

For example, to return the last 3 weeks of a time oriented zoo object, one could call last(X,'3 weeks'). Valid period.types are: secs, seconds, mins, minutes, hours, days, weeks, months, quarters, and years.

It is possible to use any frequency specification (secs, mins, days, ...) for the period.type portion of the string, even if the original data is in a higher frequency. This makes it possible to return the last '2 months' of data from an oject that has a daily periodicity.

It should be noted that it is only possible to extract data with methods equal to or less than the frequency of the original data set. Attempting otherwise will result in error.

Requesting more data than is in the original data object will produce a warning advising as such, and the object returned will simply be the original data.

Value

A subset of elements/rows of the original data.

Author(s)

Jeffrey A. Ryan

Examples

```
first(1:100)
last(1:100)

data(LakeHuron)
first(LakeHuron,10)
last(LakeHuron)

x <- xts(1:100, Sys.Date()+1:100)
first(x, 10)
first(x, '1 day')
```

```
first(x, '4 days')
first(x, 'month')
last(x, '2 months')
last(x, '6 weeks')
```

`firstof` *Create a POSIXct Object*

Description

Enable fast creation of time stamps corresponding to the first or last observation in a specified time period.

Usage

```
firstof(year = 1970, month = 1, day = 1, hour = 0, min = 0, sec = 0, tz = "")
```

Arguments

`year,month,day`	numerical values to specify a day
`hour,min,sec`	numerical vaues to specify time within a day
`tz`	timezone used for conversion

Details

A wrapper to the R function ISOdatetime with defaults corresponding to the first or last possible time in a given period.

Value

An object of class POSIXct.

Author(s)

Jeffrey A. Ryan

See Also

`ISOdatetime`

Examples

```
firstof(2000)
firstof(2005,01,01)

lastof(2007)
lastof(2007,10)
```

index.xts *Get and Replace the Class of an xts Index*

Description

Functions to get and replace an xts object's index values and it's components.

Usage

```
## S3 method for class 'xts'
index(x, ...)
## S3 replacement method for class 'xts'
index(x) <- value

.index(x, ...)
.index(x) <- value

convertIndex(x, value)

# date/time component extraction
.indexsec(x)
.indexmin(x)
.indexhour(x)

.indexDate(x)
.indexday(x)
.indexwday(x)
.indexmday(x)

.indexweek(x)
.indexmon(x)
.indexyear(x)
.indexyday(x)

.indexisdst(x)
```

Arguments

x	an xts object
value	new index value
...	arguments passed to other methods

Details

Internally, an xts object's index is a *numeric* value corresponding to seconds since the epoch in the UTC timezone. The .index and .index<- functions get and replace the internal *numeric* value of

the index, respectively. These functions are primarily for internal use, but are exported because they may be useful for users.

The `index` and `index<-` methods get and replace the xts object's index, respectively. The replacement method also updates the `tclass` and `tzone` of the index to match the class and timezone of the new index, respectively. The `index` method converts the index to the class specified by the `tclass` attribute and with the timezone specified by the `tzone` attribute before returning the index values to the user.

The `.indexXXX` functions extract time components (similar to `POSIXlt` components) from the internal time index:

`.indexsec` 0 - 61: seconds of the minute (local time)

`.indexmin` 0 - 59: minutes of the hour (local time)

`.indexhour` 0 - 23: hours of the day (local time)

`.indexDate` date as seconds since the epoch (UTC *not local time*

`.indexday` date as seconds since the epoch (UTC *not local time*

`.indexwday` 0 - 6: day of the week (Sunday - Saturday, local time)

`.indexmday` 1 - 31: day of the month (local time)

`.indexweek` weeks since the epoch (UTC *not local time*

`.indexmon` 0 - 11: month of the year (local time)

`.indexyear` years since 1900 (local time)

`.indexyday` 0 - 365: day of the year (local time, 365 only in leap years)

`.indexisdst` 1, 0, -1: Daylight Saving Time flag. Positive if Daylight Saving Time is in effect, zero if not, negative if unknown.

Changes in timezone, index class, and index format internal structure, by **xts** version:

Version 0.12.0: The `.indexTZ`, `.indexCLASS` and `.indexFORMAT` attributes are no longer stored on xts objects, only on the index itself.

The `indexTZ`, `indexClass`, and `indexFormat` functions (and their respective replacement methods) are deprecated in favor of their respective `tzone`, `tclass`, and `tformat` versions. The previous versions will throw a warning that they're deprecated, but they will continue to work. There are no plans to remove them or have them throw an error. Ever.

The latter versions are careful to look for the old attributes on the xts object, in case they're ever called on an xts object that was created prior to the attributes being added to the index itself.

There are options to throw a warning if there is no `tzone` or `tclass` attribute on the index, even if there may be one on the xts object. This gives the user a way to know if an xts object should be updated to use the new structure.

You can enable the warnings via: `options(xts.warn.index.missing.tzone = TRUE, xts.warn.index.missing.tc = TRUE)` You can identify xts objects with the old structure by printing them. Then you can update them to the new structure using `x <- as.xts(x)`.

Version 0.9.8: The index timezone is now set to "UTC" for time classes that do not have any intraday component (e.g. days, months, quarters). Previously the timezone was blank, which meant "local time" as determined by R and the OS.

Version 0.9.2: There are new get/set methods for the timezone, index class, and index format attributes: `tzone` and, `tzone<-`, `tclass` and `tclass<-`, and `tformat` and `tformat<-`.

These new functions are aliases to their `indexTZ`, `indexClass`, and `indexFormat` counterparts.

Version 0.7.5: The timezone, index class, and index format were added as attributes to the index itself, as `tzone`, `tclass`, and `tformat`, respectively. This is in order to remove those three attributes from the xts object, so they're only on the index itself.

The `indexTZ`, `indexClass`, and `indexFormat` functions (and their respective replacement methods) will continue to work as in prior **xts** versions. The attributes on the index take priority over their respective counterparts that may be on the xts object.

Versions 0.6.4 and prior: Objects track their timezone and index class in their `.indexTZ` and `.indexCLASS` attributes, respectively.

Author(s)

Jeffrey A. Ryan

See Also

`tformat` describes how the index values are formatted when printed, `tclass` provides details how **xts** handles the class of the index, and `tzone` has more information about the index timezone settings.

Examples

```
x <- timeBasedSeq('2010-01-01/2010-01-01 12:00/H')
x <- xts(seq_along(x), x)

# the index values, converted to 'tclass' (POSIXct in this case)
index(x)
class(index(x))  # POSIXct
tclass(x)        # POSIXct

# the internal numeric index
.index(x)
# add 1 hour (3600 seconds) to the numeric index
.index(x) <- index(x) + 3600
index(x)

y <- timeBasedSeq('2010-01-01/2010-01-02 12:00')
y <- xts(seq_along(y), y)

# Select all observations in the first 6 and last 3 minutes of the
# 8th and 15th hours on each day
y[.indexhour(y) %in% c(8, 15) & .indexmin(y) %in% c(0:5, 57:59)]

i <- 0:60000
focal_date <- as.numeric(as.POSIXct("2018-02-01", tz = "UTC"))
y <- .xts(i, c(focal_date + i * 15), tz = "UTC", dimnames = list(NULL, "value"))
```

```
# Select all observations for the first minute of each hour
y[.indexmin(y) == 0]

# Select all observations on Monday
mon <- y[.indexwday(y) == 1]
head(mon)
tail(mon)
unique(weekdays(index(mon))) # check

# Disjoint time of day selections

# Select all observations between 08:30 and 08:59:59.9999  or between 12:00 and 12:14:59.99999:
y[.indexhour(y) == 8 & .indexmin(y) >= 30 | .indexhour(y) == 12 & .indexmin(x) %in% 0:14]

### Compound selections

# Select all observations for Wednesdays or Fridays between 9am and 4pm (exclusive of 4pm):
y[.indexwday(y) %in% c(3, 5) & (.indexhour(y) %in%  c(9:15))]

# Select all observations on Monday between 8:59:45 and 09:04:30:

y[.indexwday(y) == 1 & (.indexhour(y) == 8 & .indexmin(y) == 59 & .indexsec(y) >= 45 |
                        .indexhour(y) == 9 &
                        (.indexmin(y) <  4 | .indexmin(y) ==  4 & .indexsec(y) <= 30))]

i <- 0:30000
u <- .xts(i, c(focal_date + i * 1800), tz = "UTC", dimnames = list(NULL, "value"))

# Select all observations for January or February:
u[.indexmon(u) %in% c(0, 1)]

# Select all data for the 28th to 31st of each month, excluding any Fridays:
u[.indexmday(u) %in% 28:31 & .indexwday(u) != 5]

# Subset by week since origin
unique(.indexweek(u))
origin <- xts(1, as.POSIXct("1970-01-01"))
unique(.indexweek(origin))

# Select all observations in weeks 2515 to 2517.
u2 <- u[.indexweek(u) %in% 2515:2517]
head(u2); tail(u2)

# Select all observations after 12pm for day 50 and 51 in each year
u[.indexyday(u) %in% 50:51 & .indexhour(u) >= 12]
```

---

isOrdered *Check If A Vector Is Ordered*

Description

Performs check to determine if a vector is strictly increasing, strictly decreasing, not decreasing, or not increasing.

Usage

```
isOrdered(x, increasing = TRUE, strictly = TRUE)
```

Arguments

x	a numeric vector
increasing	test for increasing/decreasing values
strictly	are duplicates OK

Details

Designed for internal use with **xts**, this provides highly optimized tests for ordering.

Value

Logical

Author(s)

Jeffrey A. Ryan

See Also

is.unsorted

Examples

```
# strictly increasing
isOrdered(1:10, increasing=TRUE)
isOrdered(1:10, increasing=FALSE)
isOrdered(c(1,1:10), increasing=TRUE)
isOrdered(c(1,1:10), increasing=TRUE, strictly=FALSE)

# decreasing
isOrdered(10:1, increasing=TRUE)
isOrdered(10:1, increasing=FALSE)
```

make.index.unique *Force Time Values To Be Unique*

Description

A generic function to force sorted time vectors to be unique. Useful for high-frequency time-series where original time-stamps may have identical values. For the case of xts objects, the default eps is set to ten microseconds. In practice this advances each subsequent identical time by eps over the previous (possibly also advanced) value.

Usage

```
make.index.unique(x, eps = 1e-06, drop=FALSE, fromLast=FALSE, ...)

make.time.unique(x, eps = 1e-06, drop=FALSE, fromLast=FALSE, ...)
```

Arguments

x	An xts object, or POSIXct vector.
eps	value to add to force uniqueness.
drop	drop duplicates instead of adjusting by eps
fromLast	if drop=TRUE, fromLast controls which duplicated times are dropped. If fromLast=FALSE, the earliest observation with an identical timestamp is kept with subsequent observations dropped.
...	unused

Details

The returned time-series object will have new time-stamps so that `isOrdered( .index(x) )` evaluates to TRUE.

Value

A modified version of x.

Note

Incoming values must be pre-sorted, and no check is done to make sure that this is the case. If the index values are of storage.mode 'integer', they will be coerced to 'double' if drop=FALSE.

Author(s)

Jeffrey A. Ryan

See Also

align.time

Examples

```
ds <- options(digits.secs=6) # so we can see the change

x <- xts(1:10, as.POSIXct("2011-01-21") + c(1,1,1,2:8)/1e3)
x
make.index.unique(x)

options(ds)
```

`merge.xts` *Merge xts Objects*

Description

Used to perform merge operation on `xts` objects by *time* (index). Given the inherent ordered nature of `xts` time-series, a merge-join style merge allows for optimally efficient joins.

Usage

```
## S3 method for class 'xts'
merge(...,
      all = TRUE,
      fill = NA,
      suffixes = NULL,
      join = "outer",
      retside = TRUE,
      retclass = "xts",
      tzone = NULL,
      drop=NULL,
      check.names=NULL)
```

Arguments

`...`	one or more xts objects, or objects coercible to class xts
`all`	a logical vector indicating merge type
`fill`	values to be used for missing elements
`suffixes`	to be added to merged column names
`join`	type of database join
`retside`	which side of the merged object should be returned (2-case only)
`retclass`	object to return
`tzone`	time zone of merged object
`drop`	not currently used
`check.names`	not currently used

Details

This is an xts method compatible with merge.zoo, as xts extends zoo. That documentation should also be referenced. Difference are noted where applicable.

Implemented almost entirely in custom C-level code, it is possible using either the `all` argument or the `join` argument to implement all common database join operations along the to-be-merged objects time-index: 'outer' (full outer - all rows), 'inner' (only rows with common indexes), 'left' (all rows in the left object, and those that match in the right), and 'right' (all rows in the right object, and those that match in the left).

The above join types can also be expressed as a vector of logical values passed to `all`. c(TRUE,TRUE) or TRUE for 'join="outer"', c(FALSE,FALSE) or FALSE for 'join="inner"', c(TRUE, FALSE) for 'join="left"', and c(FALSE,TRUE) for 'join="right"'.

Note that the `all` and `join` arguments imply a two case scenario. For merging more than two objects, they will simply fall back to a full outer or full inner join, depending on the first position of all, as left and right can be ambiguous with respect to sides.

To do something along the lines of merge.zoo's method of joining based on an all argument of the same length of the arguments to join, see the example.

The resultant object will have the timezone of the leftmost argument if available. Use `tzone` to override.

If `retclass` is `NULL`, the joined objects will be split and reassigned silently back to the original environment they are called from. This is for backward compatibility with zoo, though unused by xts.

If `retclass` is `FALSE` the object will be stripped of its class attribute. This is for internal use.

Value

A new `xts` object containing the appropriate elements of the objects passed in to be merged.

Note

This is a highly optimized merge, specifically designed for ordered data. The only supported merging is based on the underlying time index.

Author(s)

Jeffrey A. Ryan

References

Merge Join Discussion: https://blogs.msdn.microsoft.com/craigfr/2006/08/03/merge-join/

Examples

```
(x <- xts(4:10, Sys.Date()+4:10))
(y <- xts(1:6, Sys.Date()+1:6))

merge(x,y)
merge(x,y, join='inner')
```

```
merge(x,y, join='left')
merge(x,y, join='right')

merge.zoo(zoo(x),zoo(y),zoo(x), all=c(TRUE, FALSE, TRUE))
merge(merge(x,x),y,join='left')[,c(1,3,2)]

# zero-width objects (only index values) can be used
xi <- xts( , index(x))
merge(y, xi)
```

`na.locf.xts` *Last Observation Carried Forward*

Description

xts method replace 'NA' with most recent non-'NA'

Usage

```
## S3 method for class 'xts'
na.locf(object, na.rm = FALSE, fromLast = FALSE, maxgap=Inf, ...)
```

Arguments

`object`	an xts object
`na.rm`	logical. Should leading/trailing 'NA"s be removed? The default for xts `FALSE` is different than the default S3 method in the **zoo** package.
`fromLast`	logical. Cause observations to be carried backward rather than forward. Default is `FALSE`.
`maxgap`	runs of more than 'maxgap' will retain 'NA's after the maximum gap specified. See `na.locf` in the zoo package.
`...`	unused

Details

This is the **xts** method for the S3 generic `na.locf`. The primary difference to note is that after the 'NA' fill action is carried out, the default it to leave trailing or leading 'NA"s in place. This is different than **zoo** behavior.

Value

See the documentation in zoo.

Author(s)

Jeffrey A. Ryan

References

'zoo'

Examples

```
x <- xts(1:10, Sys.Date()+1:10)
x[c(1,2,5,9,10)] <- NA

x
na.locf(x)
na.locf(x, fromLast=TRUE)
na.locf(x, na.rm=TRUE, fromLast=TRUE)
```

---

ndays *Number of Periods in Data*

Description

Calculate the number of specified periods in a given time series like data object.

Usage

```
nseconds(x)
nminutes(x)
nhours(x)
ndays(x)
nweeks(x)
nmonths(x)
nquarters(x)
nyears(x)
```

Arguments

x	A time-based object

Details

Essentially a wrapper to `endpoints` with the appropriate period specified; the resulting value derived from counting the endpoints

As a compromise between simplicity and accuracy, the results will always round up to the nearest complete period. So n**** - 1 will return the completed periods.

For finer grain detail one should call a higher frequency n**** function.

An alternative summary can be found with `periodicity` and `unclass(periodicity(x))`.

Value

The number of observations for the period type specified

Author(s)

Jeffrey A. Ryan

See Also

endpoints

Examples

```
## Not run:
getSymbols("QQQQ")

ndays(QQQQ)
nweeks(QQQQ)

## End(Not run)
```

---

period.apply *Apply Function Over Specified Interval*

Description

Apply a specified function to data over intervals specified by INDEX. The intervals are defined as the observations from INDEX[k]+1 to INDEX[k+1], for k = 1:(length(INDEX)-1).

Usage

```
period.apply(x, INDEX, FUN, ...)
```

Arguments

x	The data that FUN will be applied to.
INDEX	A numeric vector of index breakpoint locations. The vector should begin with 0 and end with NROW(x).
FUN	A function to apply to each interval in x.
...	Additional arguments for FUN.

Details

Similar to the rest of the apply family, period.apply() calculates the specified function's value over a subset of data. The primary difference is that period.apply() applies the function to non-overlapping intervals of a vector or matrix.

Useful for applying functions over an entire data object by any non-overlapping intervals. For example, when INDEX is the result of a call to endpoints().

period.apply() checks that INDEX is sorted, unique, starts with 0, and ends with NROW(x). All those conditions are true of vectors returned by endpoints().

Value

An object with `length(INDEX) - 1` observations (assuming `INDEX` starts with 0 and ends with `NROW(x)`).

Note

When `FUN = mean` the results will contain one column for every column in the input, which is different from other math functions (e.g. `median`, `sum`, `prod`, `sd`, etc.).

`FUN = mean` works by column because the default method `stats::mean` used to work by column for matrices and data.frames. R Core changed the behavior of `mean` to always return one column in order to be consistent with the other math functions. This broke some **xts** dependencies and `mean.xts` was created to maintain the original behavior.

Using `FUN = mean` will print a message that describes this inconsistency. To avoid the message and confusion, use `FUN = colMeans` to calculate means by column and use `FUN = function(x) mean` to calculate one mean for all the data. Set `options(xts.message.period.apply.mean = FALSE)` to suppress this message.

Author(s)

Jeffrey A. Ryan, Joshua M. Ulrich

See Also

`endpoints` `apply.monthly`

Examples

```
zoo.data <- zoo(rnorm(31)+10,as.Date(13514:13744,origin="1970-01-01"))
ep <- endpoints(zoo.data,'weeks')
period.apply(zoo.data, INDEX=ep, FUN=function(x) colMeans(x))
period.apply(zoo.data, INDEX=ep, FUN=colMeans)                  #same

period.apply(letters,c(0,5,7,26), paste0)
```

`period.max` *Calculate Max By Period*

Description

Calculate a maximum for each period of INDEX. Essentially a rolling application of maximum over a series of non-overlapping sections.

Usage

```
period.max(x, INDEX)
```

Arguments

x	a univariate data object
INDEX	a numeric vector of endpoints to calculate maximum on

Details

Used to calculate a maximum per period given an arbitrary index of sections to be calculated over. This is an optimized function for maximum. There are additional optimized versions for min, sum, and prod.

For xts-coercible objects, an appropriate INDEX can be derived from a call to 'endpoints'.

Value

An xts or zoo object of maximums, indexed by the period endpoints.

Author(s)

Jeffrey A. Ryan

See Also

endpoints, period.sum, period.min, period.prod

Examples

```
period.max(c(1,1,4,2,2,6,7,8,-1,20),c(0,3,5,8,10))

data(sample_matrix)
period.max(sample_matrix[,1],endpoints(sample_matrix))
period.max(as.xts(sample_matrix)[,1],endpoints(sample_matrix))
```

---

period.min *Calculate Min By Period*

Description

Calculate a minimum for each period of INDEX. Essentially a rolling application of minimum over a series of non-overlapping sections.

Usage

```
period.min(x, INDEX)
```

Arguments

x	a univariate data object
INDEX	a numeric vector of endpoints to calculate maximum on

Details

Used to calculate a minimum per period given an arbitrary index of sections to be calculated over. This is an optimized function for minimum. There are additional optimized versions for max, sum, and prod.

For xts-coercible objects, an appropriate INDEX can be derived from a call to `endpoints`.

Value

An xts or zoo object of minimums, indexed by the period endpoints.

Author(s)

Jeffrey A. Ryan

See Also

`endpoints`, `period.sum`, `period.max`, `period.prod`

Examples

```
period.min(c(1,1,4,2,2,6,7,8,-1,20),c(0,3,5,8,10))

data(sample_matrix)
period.min(sample_matrix[,1],endpoints(sample_matrix))
period.min(as.xts(sample_matrix)[,1],endpoints(sample_matrix))
```

---

period.prod *Calculate Product By Period*

Description

Calculate a product for each period of INDEX. Essentially a rolling application of prod over a series of non-overlapping sections.

Usage

```
period.prod(x, INDEX)
```

Arguments

x	a univariate data object
INDEX	a vector of breakpoints to calculate product on

Details

Used to calculate a product per period given an arbitrary index of sections to be calculated over. This is an optimized function for product. There are additionally optimized versions for min, max, and sum.

For xts-coercible objects, an appropriate INDEX can be derived from a call to `endpoints`.

Value

An xts or zoo object of products, indexed by the period endpoints.

Author(s)

Jeffrey A. Ryan

See Also

endpoints, period.sum, period.min, period.max

Examples

```
period.prod(c(1,1,4,2,2,6,7,8,-1,20),c(0,3,5,8,10))

data(sample_matrix)
period.prod(sample_matrix[,1],endpoints(sample_matrix))
period.prod(as.xts(sample_matrix)[,1],endpoints(sample_matrix))
```

---

period.sum *Calculate Sum By Period*

Description

Calculate a sum for each period of INDEX. Essentially a rolling application of sum over a series of non-overlapping sections.

Usage

```
period.sum(x, INDEX)
```

Arguments

x	a univariate data object
INDEX	a numeric vector of endpoints to calculate sum on

Details

Used to calculate a sum per period given an arbitrary index of sections to be calculated over. This is an optimized function for sum. There are additionally optimized versions for min, max, and prod.

For xts-coercible objects, an appropriate INDEX can be derived from a call to endpoints.

Value

An xts or zoo object of sums, indexed by the period endpoints.

Author(s)

Jeffrey A. Ryan

See Also

endpoints, period.max, period.min, period.prod

Examples

```
period.sum(c(1,1,4,2,2,6,7,8,-1,20),c(0,3,5,8,10))

data(sample_matrix)
period.sum(sample_matrix[,1],endpoints(sample_matrix))
period.sum(as.xts(sample_matrix)[,1],endpoints(sample_matrix))
```

---

periodicity *Approximate Series Periodicity*

Description

Estimate the periodicity of a time-series-like object by calculating the median time between observations in days.

Usage

```
periodicity(x, ...)
```

Arguments

x	time-series-like object
...	unused

Details

A simple wrapper to quickly estimate the periodicity of a given data. Returning an object of type periodicity.

This calculates the median number of days between observations as a difftime object, the numerical difference, the units of measurement, and the derived scale of the data as a string.

The time index currently must be of either Date or POSIX class, or coercible to such.

The only list item of note is the scale. This is an estimate of the periodicity of the data in common terms - e.g. 7 day daily data is best described as 'weekly', and would be returned as such.

Possible scale values are:

'minute','hourly', 'daily','weekly', 'monthly','quarterly', and 'yearly'.

Value

An object containing a list containing the difftime object, frequency, units, and suitable scale.

Note

This function is only a *good estimate* for the underlying periodicity. If the series is too short, or has *no* real periodicity, the return values will obviously be wrong. That said, it is quite robust and used internally within **xts**.

Author(s)

Jeffrey A. Ryan

See Also

difftime

Examples

```
zoo.ts <- zoo(rnorm(231),as.Date(13514:13744,origin="1970-01-01"))
periodicity(zoo.ts)
```

---

plot.xts *Plotting xts Objects*

Description

Plotting for xts objects.

Usage

```
## S3 method for class 'xts'
plot(x, y = NULL, ..., subset = "",
  panels = NULL, multi.panel = FALSE, col = 1:8, up.col = NULL,
  dn.col = NULL, bg = "#FFFFFF", type = "l", lty = 1, lwd = 2, lend = 1,
  main = deparse(substitute(x)), main.timespan = TRUE, observation.based = FALSE,
  log = FALSE, ylim = NULL, yaxis.same = TRUE, yaxis.left = TRUE, yaxis.right = TRUE,
  yaxis.ticks = 5, major.ticks = "auto", minor.ticks = NULL,
  grid.ticks.on = "auto", grid.ticks.lwd = 1, grid.ticks.lty = 1,
  grid.col = "darkgray", labels.col = "#333333", format.labels = TRUE,
  grid2 = "#F5F5F5", legend.loc = NULL, extend.xaxis = FALSE)
## S3 method for class 'xts'
lines(x, ..., main = "", on = 0, col = NULL, type = "l",
  lty = 1, lwd = 1, pch = 1)
## S3 method for class 'xts'
points(x, ..., main = "", on = 0, col = NULL, pch = 1)
```

Arguments

x	xts object
y	NULL, not used
...	any passthrough graphical arguments for lines and points
subset	character vector of length one of the subset range using subsetting as in xts
panels	character vector of expressions to plot as panels
multi.panel	TRUE/FALSE or an integer less than or equal to the number of columns in the data set. If TRUE, each column of the data is plotted in a separate panel. For example, if multi.panel = 2, then the data will be plotted in groups of 2 columns and each group is plotted in a separate panel.
col	color palette to use, set by default to rational choices
up.col	color for positive bars if type="h"
dn.col	color for negative bars if type="h"
bg	background color of plotting area, same as in par
type	the type of plot to be drawn, same as in plot
lty	set the line type, same as in par
lwd	set the line width, same as in par
lend	set the line end style, same as in par
main	main title
main.timespan	include the timespan of the series on the plot? (default TRUE)
observation.based	TRUE/FALSE (default FALSE). If TRUE, the x-axis is drawn based on observations in the data. If FALSE, the x-axis is drawn based on the time index of the data.
log	TRUE/FALSE (default FALSE). If TRUE, the y-axis is drawn in log-scale
ylim	the range of the y axis
yaxis.same	TRUE/FALSE. If TRUE, the y axis is drawn with the same ylim for multiple panels
yaxis.left	if TRUE, draws the y axis on the left
yaxis.right	if TRUE, draws the y axis on the right
yaxis.ticks	desired number of y axis grid lines. The actual number of grid lines is determined by the n argument to pretty.
major.ticks	period that specifies where tick marks and labels will be drawn on the x-axis. See Details for possible values.
minor.ticks	period that specifies where minor ticks on will be drawn on the x-axis. If NULL, minor ticks are not drawn. See Details for possible values.
grid.ticks.on	period that specifies where vertical grid lines will be drawn. See Details for possible values.
grid.ticks.lwd	line width of the grid
grid.ticks.lty	line type of the grid

grid.col	color of the grid
labels.col	color of the axis labels
format.labels	label format to draw lower frequency x-axis ticks and labels passed to axTicksByTime
grid2	color for secondary x axis grid
legend.loc	places a legend into one of nine locations on the chart: bottomright, bottom, bottomleft, left, topleft, top, topright, right, or center. Default NULL does not draw a legend.
pch	the plotting character to use, same as in par.
on	panel number to draw on. A new panel will be drawn if on=NA. The default, on=0, will add to the active panel. The active panel is defined as the panel on which the most recent action was performed. Note that only the first element of on is checked for the default behavior to add to the last active panel.
extend.xaxis	TRUE/FALSE (default FALSE). If TRUE, extend the x-axis before and/or after the plot's existing time index range, so all of of the time index values of the new series are included in the plot.

Details

Possible values for arguments major.ticks, minor.ticks, and grid.ticks.on include 'auto', 'minute', 'hours', 'days', 'weeks', 'months', 'quarters', and 'years'. The default is 'auto', which attempts to determine sensible locations from the periodicity and locations of observations. The other values are based on the possible values for the ticks.on argument of axTicksByTime.

Author(s)

Ross Bennett

References

based on chart_Series in the quantmod package by Jeffrey A. Ryan

See Also

addSeries, addPanel

Examples

```
## Not run:
data(sample_matrix)
sample.xts <- as.xts(sample_matrix)

# plot the Close
plot(sample.xts[,"Close"])

# plot a subset of the data
plot(sample.xts[,"Close"], subset="2007-04-01/2007-06-31")

# function to compute simple returns
```

```
simple.ret <- function(x, col.name){
  x[,col.name] / lag(x[,col.name]) - 1
}

# plot the close and add a panel with the simple returns
plot(sample.xts[,"Close"])
R <- simple.ret(sample.xts, "Close")
lines(R, type="h", on=NA)

# add the 50 period simple moving average to panel 1 of the plot
library(TTR)
lines(SMA(sample.xts[,"Close"], n = 50), on=1, col="blue")

# add month end points to the chart
points(sample.xts[endpoints(sample.xts[,"Close"], on = "months"), "Close"],
       col="red", pch=17, on=1)

# add legend to panel 1
addLegend("topright", on=1,
          legend.names = c("Close", "SMA(50)"),
          lty=c(1, 1), lwd=c(2, 1),
          col=c("black", "blue", "red"))

## End(Not run)
```

`print.xts` *Print An xts Time-Series Object*

Description

Method for printing an extensible time-series object.

Usage

```
## S3 method for class 'xts'
print(x, fmt, ..., show.rows = 10, max.rows = 100)
```

Arguments

`x`	An xts object
`fmt`	Passed to `coredata` to format the time index
`...`	Arguments passed to other methods
`show.rows`	The number of first and last rows to print if the number of rows is truncated (default 10, or `getOption("xts.print.show.rows")`)
`max.rows`	The output will contain at most `max.rows` rows before being truncated (default 100, or `getOption("xts.print.max.rows")`)

Value

Returns x invisibly.

Author(s)

Joshua M. Ulrich

Examples

```
data(sample_matrix)
sample.xts <- as.xts(sample_matrix)

# output is truncated and shows first and last 10 observations
print(sample.xts)

# show the first and last 5 observations
print(sample.xts, show.rows = 5)
```

---

rbind.xts *Concatenate Two or More xts Objects by Row*

Description

Concatenate or bind by row two or more xts objects along a time-based index.

Usage

```
## S3 method for class 'xts'
c(...)

## S3 method for class 'xts'
rbind(..., deparse.level = 1)
```

Arguments

...	objects to bind
deparse.level	not implemented

Details

Implemented in C, these functions bind xts objects by row, resulting in another xts object

There may be non-unique index values in either the original series, or the resultant series.

Identical indexed series are bound in the order or the arguments passed to rbind. See examples.

All objects must have the same number of columns, as well as be xts objects or coercible to such.

rbind and c are aliases.

For traditional merge operations, see merge.xts and cbind.xts.

Value

An `xts` object with one row per row for each object concatenated.

Note

This differs from rbind.zoo in that non-unique index values are allowed, in addition to the completely different algorithms used internally.

All operations may not behave as expected on objects with non-unique indices. You have been warned.

`rbind` is a .Primitive function in R. As such method dispatch occurs at the C-level, and may not be consistent with expectations. See the details section of the base function, and if needed call rbind.xts directly to avoid dispatch ambiguity.

Author(s)

Jeffrey A. Ryan

See Also

`merge.xts` `rbind`

Examples

```
x <- xts(1:10, Sys.Date()+1:10)
str(x)

merge(x,x)
rbind(x,x)
rbind(x[1:5],x[6:10])

c(x,x)

# this also works on non-unique index values
x <- xts(rep(1,5), Sys.Date()+c(1,2,2,2,3))
y <- xts(rep(2,3), Sys.Date()+c(1,2,3))

# overlapping indexes are appended
rbind(x,y)
rbind(y,x)
```

`sample_matrix` *Sample Data Matrix For xts Example and Unit Testing*

Description

Simulated 180 observations on 4 variables.

Usage

```
data(sample_matrix)
```

Format

```
  The format is:
 num [1:180, 1:4] 50.0 50.2 50.4 50.4 50.2 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:180] "2007-01-02" "2007-01-03" "2007-01-04" "2007-01-05" ...
  ..$ : chr [1:4] "Open" "High" "Low" "Close"
```

Examples

```
data(.sample.matrix)
```

---

split.xts *Divide into Groups by Time*

Description

Creates a list of xts objects split along time periods.

Usage

```
## S3 method for class 'xts'
split(x, f = "months", drop=FALSE, k = 1, ...)
```

Arguments

x	an xts object
f	a 'character' vector describing the period to split by
drop	ignored by split.xts
k	number of periods to aggregate into each split. See Details.
...	further args to non-xts method

Details

A quick way to break up a large xts object by standard time periods; e.g. 'months', 'quarters', etc.

`endpoints` is used to find the start and end of each period (or k-periods). See that function for valid arguments.

If `f` is not a character vector, the NextMethod is called, which would in turn dispatch to the split.zoo method.

Value

A list of xts objects.

Note

aggregate.zoo would be more flexible, though not as fast for xts objects.

Author(s)

Jeffrey A. Ryan

See Also

endpoints, split.zoo, aggregate.zoo

Examples

```
data(sample_matrix)
x <- as.xts(sample_matrix)


split(x)
split(x, f="weeks")
split(x, f="weeks", k=4)
```

---

tclass *Get or Replace the Class of an xts Object's Index*

Description

Generic functions to get or replace the class of an xts object's index.

Usage

```
tclass(x, ...)
tclass(x) <- value

## S3 method for class 'xts'
tclass(x, ...)
## S3 replacement method for class 'xts'
tclass(x) <- value

##### The functions below are DEPRECATED #####
indexClass(x)
indexClass(x) <- value
```

Arguments

x	an xts object
value	new index class (see Details for valid values)
...	arguments passed to other methods

Details

Internally, an xts object's index is a *numeric* value corresponding to seconds since the epoch in the UTC timezone. The index class is stored as the `tclass` attribute on the internal index. This is used to convert the internal index values to the desired class when the `index` function is called.

The `tclass` function retrieves the class of the internal index, and the `tclass<-` function sets it. The specified value for `tclass<-` must be one of the following character strings: `"Date"`, `"POSIXct"`, `"chron"`, `"yearmon"`, `"yearqtr"`, or `"timeDate"`.

Value

A vector containing the class of the object's index.

Note

Both `indexClass` and `indexClass<-` are deprecated in favor of `tclass` and `tclass<-`, respectively.

Replacing the `tclass` *does not* change the values of the internal index. See the examples.

Author(s)

Jeffrey A. Ryan

See Also

`index` has more information on the xts index, `tformat` details how the index values are formatted when printed, `tzone` has more information about the index timezone settings.

The following help pages describe the characteristics of the valid index classes: `POSIXct`, `Date`, `chron`, `yearmon`, `yearqtr`, `timeDate`.

Examples

```
x <- timeBasedSeq('2010-01-01/2010-01-02 12:00')
x <- xts(seq_along(x), x)


y <- timeBasedSeq('2010-01-01/2010-01-03 12:00/H')
y <- xts(seq_along(y), y, tzone = "America/New_York")

# Changing the tclass does not change the internal index values, but it
# does change how the index is printed!
head(y)    # the index has times
.index(y)
tclass(y) <- "Date"
head(y)    # the index prints without times, but
.index(y)  # the internal index is not changed!
```

`tformat` *Get or Replace the Format of an xts Object's Index*

Description

Generic functions to get or replace the format that determines how an xts object's index is printed.

Usage

```
tformat(x, ...)
tformat(x) <- value

## S3 method for class 'xts'
tformat(x, ...)
## S3 replacement method for class 'xts'
tformat(x) <- value

##### The functions below are DEPRECATED #####
indexFormat(x)
indexFormat(x) <- value
```

Arguments

`x`	an `xts` object
`value`	new index format string (see Details for valid values)
`...`	arguments passed to other methods

Details

Valid values for the `value` argument are the same as specified in the *Details* section of `strptime`.

An xts object's `tformat` is `NULL` by default, so the index will be formatted according to its `tclass` (e.g. `Date`, `POSIXct`, `timeDate`, `yearmon`, etc.).

`tformat` only changes how the index is *printed* and how the row names are formatted when xts objects are converted to other classes (e.g. `matrix` or `data.frame`. It does not affect the internal index in any way.

Value

A vector containing the format for the object's index.

Note

Both `indexFormat` and `indexFormat<-` are deprecated in favor of `tformat` and `tformat<-`, respectively.

Author(s)

Jeffrey A. Ryan

See Also

index has more information on the xts index, tclass details how **xts** handles the class of the index, tzone has more information about the index timezone settings.

Examples

```
x <- timeBasedSeq('2010-01-01/2010-01-02 12:00')
x <- xts(seq_along(x), x)

# set a custom index format
head(x)
tformat(x) <- "%Y-%b-%d %H:%M:%OS3"
head(x)
```

---

timeBased *Check if Class is Time-Based*

Description

Used to verify that the object is one of the known time-based classes in R.

Usage

```
is.timeBased(x)
timeBased(x)
```

Arguments

x	object to test

Details

Current time-based objects supported are Date, POSIXct, chron, yearmon, yearqtr, and timeDate.

Value

Logical

Author(s)

Jeffrey A. Ryan

Examples

```
timeBased(Sys.time())
timeBased(Sys.Date())

timeBased(200701)
```

`timeBasedSeq` *Create a Sequence or Range of Times*

Description

A function to create a vector of time-based objects suitable for indexing an *xts* object, given a string conforming to the ISO 8601 time and date standard for range-based specification. The resultant series can be of any class supported by *xts*, including POSIXct, Date, chron, timeDate, yearmon, and yearqtr.

`timeBasedRange` creates a vector of length 1 or 2 as seconds since the epoch (1970-01-01) for use internally.

Usage

```
timeBasedSeq(x, retclass = NULL, length.out = NULL)

timeBasedRange(x, ...)
```

Arguments

`x`	a string representing the time-date range desired
`retclass`	the return class desired
`length.out`	passed to `seq` internally
`...`	unused

Details

Designed to provide uniform creation of valid time-based objects for use within *xts*, the interface conforms (mostly) to the ISO recommended format for specifying ranges.

In general, the format is a string specifying a time and/or date *from*, *to*, and optionally *by* delineated by either '"/"' or '"::"'.

The first argument need not be quoted, as it is converted internally if need be.

The general form is *from/to/by* or *from::to::by*, where *to* and *by* are optional if the length.out arg is specified.

The `from` and `to` elements of the string must be left-specified with respect to the standard *CCYYMMDD HHMMSS* form. All dates-times specified will be set to either the earliest point (from) or the latest (to), given the level of specificity.

For example '1999' in the *from* field would set the start to the beginning of 1999. The opposite occurs in the *to* field.

The level of detail in the request is interpretted as the level of detail in the result. The maximum detail of either *from* or *to* is the basis of the sequence, unless the optional *by* element is specified, which will be covered later.

To request a yearly series, it is only necessary to use '"1999/2008"'. Alternately, one could request a monthly series (returned by default as class `yearmon`) with '"199901/2008"' or '"1999-01/2008"', or even '"1999/2008-01"'. As the level of granularity increases, so does the resultant sequence granularity - as does its length.

Using the optional third *by* field (the third delimited element to the string), will override the granularity intepretation and return the requested periodicity. The acceptable arguments include `Y` for years, `m` for months, `d` for days, `H` for hours, `M` for minutes and `S` for seconds.

Value

A sequence or range of time-based objects.

If `retclass` is `NULL`, the result is a named list of from, to, by and length.out.

Author(s)

Jeffrey A. Ryan

References

International Organization for Standardization: ISO 8601 https://www.iso.org

See Also

timeBased, xts

Examples

```
timeBasedSeq('1999/2008')
timeBasedSeq('199901/2008')
timeBasedSeq('199901/2008/d')
timeBasedSeq('20080101 0830',length=100) # 100 minutes
timeBasedSeq('20080101 083000',length=100) # 100 seconds
```

---

to.period *Convert time series data to an OHLC series*

Description

Convert an OHLC or univariate object to a specified periodicity lower than the given data object. For example, convert a daily series to a monthly series, or a monthly series to a yearly one, or a one minute series to an hourly series.

The result will contain the open and close for the given period, as well as the maximum and minimum over the new period, reflected in the new high and low, respectively.

If volume for a period was available, the new volume will also be calculated.

Usage

```
to.minutes(x,k,name,...)
to.minutes3(x,name,...)
to.minutes5(x,name,...)
to.minutes10(x,name,...)
to.minutes15(x,name,...)
to.minutes30(x,name,...)
to.hourly(x,name,...)
to.daily(x,drop.time=TRUE,name,...)

to.weekly(x,drop.time=TRUE,name,...)
to.monthly(x,indexAt='yearmon',drop.time=TRUE,name,...)
to.quarterly(x,indexAt='yearqtr',drop.time=TRUE,name,...)
to.yearly(x,drop.time=TRUE,name,...)

to.period(x,
          period = 'months',
          k = 1,
          indexAt,
          name=NULL,
          OHLC = TRUE,
          ...)
```

Arguments

x	a univariate or OHLC type time-series object
period	period to convert to. See details.
indexAt	convert final index to new class or date. See details
drop.time	remove time component of POSIX datestamp (if any)
k	number of sub periods to aggregate on (only for minutes and seconds)
name	override column names
OHLC	should an OHLC object be returned? (only OHLC=TRUE currently supported)
...	additional arguments

Details

Essentially an easy and reliable way to convert one periodicity of data into any new periodicity. It is important to note that all dates will be aligned to the *end* of each period by default - with the exception of to.monthly and to.quarterly, which index by 'yearmon' and 'yearqtr' from the **zoo** package, respectively.

Valid period character strings include: "seconds", "minutes", "hours", "days", "weeks", "months", "quarters", and "years". These are calculated internally via endpoints. See that function's help page for further details.

To adjust the final indexing style, it is possible to set indexAt to one of the following: 'yearmon', 'yearqtr', 'firstof', 'lastof', 'startof', or 'endof'. The final index will then be yearmon, yearqtr,

the first time of the period, the last time of the period, the starting time in the data for that period, or the ending time in the data for that period, respectively.

It is also possible to pass a single time series, such as a univariate exchange rate, and return an OHLC object of lower frequency - e.g. the weekly OHLC of the daily series.

Setting `drop.time` to `TRUE` (the default) will convert a series that includes a time component into one with just a date index, as the time index is often of little value in lower frequency series.

It is not possible to convert a series from a lower periodicity to a higher periodicity - e.g. weekly to daily or daily to 5 minute bars, as that would require magic.

Value

An object of the original type, with new periodicity.

Note

In order for this function to work properly on OHLC data, it is necessary that the Open, High, Low and Close columns be names as such; including the first letter capitalized and the full spelling found. Internally a call is made to reorder the data into the correct column order, and then a verification step to make sure that this ordering and naming has succeeded. All other data formats must be aggregated with functions such as `aggregate` and `period.apply`.

This method should work on almost all time-series-like objects. Including 'timeSeries', 'zoo', 'ts', and 'irts'. It is even likely to work well for other data structures - including 'data.frames' and 'matrix' objects.

Internally a call to `as.xts` converts the original `x` into the universal `xts` format, and then re-converts back to the original type.

A special note with respect to 'ts' objects. As these are strictly regular they may include `NA` values. These are stripped for aggregation purposes, though replaced before returning. This inevitably leads to many, many additional 'NA' values in the data. It is more beneficial to consider using an 'xts' object originally, or converting to one in the function call by means of `as.xts`.

Author(s)

Jeffrey A. Ryan

Examples

```
data(sample_matrix)

samplexts <- as.xts(sample_matrix)

to.monthly(samplexts)
to.monthly(sample_matrix)

str(to.monthly(samplexts))
str(to.monthly(sample_matrix))
```

tzone *Get or Replace the Timezone of an xts Object's Index*

Description

Generic functions to get or replace the timezone of an xts object's index.

Usage

```
tzone(x, ...)
tzone(x) <- value

## S3 method for class 'xts'
tzone(x, ...)
## S3 replacement method for class 'xts'
tzone(x) <- value

##### The functions below are DEPRECATED #####
indexTZ(x, ...)
indexTZ(x) <- value
```

Arguments

x	an xts object
value	a valid timezone value (see OlsonNames())
...	arguments passed to other methods

Details

Internally, an xts object's index is a *numeric* value corresponding to seconds since the epoch in the UTC timezone. When an xts object is created, all time index values are converted internally to POSIXct (which is also in seconds since the UNIX epoch), using the underlying OS conventions and the TZ environment variable. The xts() function manages timezone information as transparently as possible.

The tzone<- function *does not* change the internal index values (i.e. the index will remain the same time in the UTC timezone).

Value

A one element named vector containing the timezone of the object's index.

Note

Both indexTZ and indexTZ<- are deprecated in favor of tzone and tzone<-, respectively.

Problems may arise when an object that had been created under one timezone are used in a session using another timezone. This isn't usually a issue, but when it is a warning is given upon printing or subsetting. This warning may be suppressed by setting options(xts_check_TZ = FALSE).

Note

Both indexTZ and indexTZ<- are deprecated in favor of tzone and tzone<-, respectively.

Timezones are a difficult issue to manage. It's best to set the system TZ environment variable to "GMT" or "UTC" (via Sys.setenv(TZ = "UTC")) at the beginning of your scripts if you do not need intra-daily resolution.

Author(s)

Jeffrey A. Ryan

See Also

POSIXt index has more information on the xts index, tformat describes how the index values are formatted when printed, and tclass provides details how **xts** handles the class of the index.

Examples

```
# Date indexes always have a "UTC" timezone
x <- xts(1, Sys.Date())
tzone(x)
str(x)
print(x)

# The default 'tzone' is blank -- your machine's local timezone,
# determined by the 'TZ' environment variable.
x <- xts(1, Sys.time())
tzone(x)
str(x)

# now set 'tzone' to different values
tzone(x) <- "UTC"
str(x)

tzone(x) <- "America/Chicago"
str(x)

y <- timeBasedSeq('2010-01-01/2010-01-03 12:00/H')
y <- xts(seq_along(y), y, tzone = "America/New_York")

# Changing the tzone does not change the internal index values, but it
# does change how the index is printed!
head(y)
head(.index(y))
tzone(y) <- "Europe/London"
head(y)          # the index prints with hours, but
head(.index(y))  # the internal index is not changed!
```

window.xts *Extract time windows from an* xts *series*

Description

Method for extracting time windows from xts objects.

Usage

```
## S3 method for class 'xts'
window(x, index. = NULL, start = NULL, end = NULL, ...)
```

Arguments

x	an object.
index.	a user defined time index. This defaults to the xts index for the series via .index(x). When supplied, this is typically a subset of the dates in the full series. The index. must be a set of dates that are convertible to POSIXct. If you want fast lookups, then index. should be sorted and of class POSIXct. If an unsorted index. is passed in, window will sort it.
start	a start time. Extract xts rows where index. >= start. start may be any class that is convertible to POSIXct such as a character variable in the format 'YYYY-MM-DD'. If start is NULL then all index. dates are matched.
end	an end time. Extract xts rows where index. <= end. end must be convertible to POSIXct. If end is NULL then all index. dates are matched.
...	currently not used.

Details

The point of having window in addition to the regular subset function is to have a fast way of extracting time ranges from an xts series. In particular, this method will convert start and end to POSIXct then do a binary lookup on the internal xts index to quickly return a range of matching dates. With a user supplied index., a similarly fast invocation of findInterval is used so that large sets of sorted dates can be retrieved quickly.

Value

The matching time window is extracted.

Author(s)

Corwin Joy

See Also

subset.xts, findInterval, xts

Examples

```
## xts example
x.date <- as.Date(paste(2003, rep(1:4, 4:1), seq(1,19,2), sep = "-"))
x <- xts(matrix(rnorm(20), ncol = 2), x.date)
x

window(x, start = "2003-02-01", end = "2003-03-01")
window(x, start = as.Date("2003-02-01"), end = as.Date("2003-03-01"))
window(x, index = x.date[1:6], start = as.Date("2003-02-01"))
window(x, index = x.date[c(4, 8, 10)])

## Assign to subset
window(x, index = x.date[c(4, 8, 10)]) <- matrix(1:6, ncol = 2)
x
```

---

xts *Create Or Test For An xts Time-Series Object*

Description

Constructor function for creating an extensible time-series object.

xts is used to create an xts object from raw data inputs.

Usage

```
xts(x = NULL,
    order.by = index(x),
    frequency = NULL,
    unique = TRUE,
    tzone = Sys.getenv("TZ"),
    ...)

.xts(x = NULL,
     index,
     tclass = c("POSIXct", "POSIXt"),
     tzone = Sys.getenv("TZ"),
     check = TRUE,
     unique = FALSE,
     ...)

is.xts(x)
```

Arguments

x	an object containing the time series data
order.by	a corresponding vector of dates/times of a known time-based class. See Details.
index	a corresponding *numeric* vector specified as seconds since the UNIX epoch (1970-01-01 00:00:00.000)
frequency	numeric indicating frequency of order.by. See Details.
unique	check the index for unique timestamps?
check	check that the index is ordered?
tclass	time class to use for the index. See tclass.
tzone	time zone of the index (ignored indices without a time component, e.g. Date, yearmon, yearqtr). See tzone.
...	additional attributes to be added. See Details.

Details

An xts object extends the S3 class zoo from the package of the same name.

The xts() constructor is the preferred way to create xts objects. It performs several checks to ensure it returns a well-formed xts object. The .xts() constructor is mainly for internal use. It is more efficient than the regular xts() constructor because it doesn't perform as many validity checks. Use it with caution.

Similar to zoo objects, xts objects must have an ordered index. While zoo indexes cannot contain duplicate values, xts objects have optionally supported duplicate index elements since version 0.5-0. The xts class has one additional requirement, the index must be a time-based class. Currently supported classes include: 'Date', 'POSIXct', 'timeDate', as well as 'yearmon' and 'yearqtr' where the index values remain unique.

The uniqueness requirement was relaxed in version 0.5-0, but is still enforced by default. Setting unique = FALSE skips the uniqueness check and only ensures that the index is ordered via the isOrdered function.

As of version 0.10-0, xts no longer allows missing values in the index. This is because many xts functions expect all index values to be finite. The most important of these is merge.xts, which is used ubiquitously. Missing values in the index are usually the result of a date-time conversion error (e.g. incorrect format, non-existent time due to daylight saving time, etc). Because of how non-finite numbers are represented, a missing timestamp will always be at the end of the index (except if it is -Inf, which will be first).

Another difference from **zoo** is that xts object may carry additional attributes that may be desired in individual time-series handling. This includes the ability to augment the objects data with meta-data otherwise not cleanly attachable to a standard zoo object.

Examples of usage from finance may include the addition of data for keeping track of sources, last-update times, financial instrument descriptions or details, etc.

The idea behind xts is to offer the user the ability to utilize a standard zoo object, while providing an mechanism to customize the object's meta-data, as well as create custom methods to handle the object in a manner required by the user.

Many xts-specific methods have been written to better handle the unique aspects of xts. These include, '"["', merge, cbind, rbind, c, Ops, lag, diff, coredata, head and tail. Additionally there are xts specific methods for converting to/from R's different time-series classes.

Subsetting via "[" methods offers the ability to specify dates by range, if they are enclosed in quotes. The style borrows from python by creating ranges with a double colon "":::"" or ""/"" operator. Each side of the operator may be left blank, which would then default to the beginning and end of the data, respectively. To specify a subset of times, it is only required that the time specified be in standard ISO format, with some form of separation between the elements. The time must be 'left-filled', that is to specify a full year one needs only to provide the year, a month would require the full year and the integer of the month requested - e.g. '1999-01'. This format would extend all the way down to seconds - e.g. '1999-01-01 08:35:23'. Leading zeros are not necessary. See the examples for more detail.

Users may also extend the `xts` class to new classes to allow for method overloading.

Additional benefits derive from the use of `as.xts` and `reclass`, which allow for lossless two-way conversion between common R time-series classes and the `xts` object structure. See those functions for more detail.

Value

An S3 object of class `xts`. As it inherits and extends the zoo class, all zoo methods remain valid. Additional attributes may be assigned and extracted via `xtsAttributes`.

Note

Most users will benefit the most by using the `as.xts` and `reclass` functions to automagically handle *all* data objects as one would handle a zoo object.

Author(s)

Jeffrey A. Ryan and Joshua M. Ulrich

References

zoo:

See Also

`as.xts`, `index`, `tclass`, `tformat`, `tzone`, `xtsAttributes`

Examples

```
data(sample_matrix)
sample.xts <- as.xts(sample_matrix, descr='my new xts object')

class(sample.xts)
str(sample.xts)

head(sample.xts)  # attribute 'descr' hidden from view
attr(sample.xts,'descr')
```

```
sample.xts['2007']  # all of 2007
sample.xts['2007-03/']  # March 2007 to the end of the data set
sample.xts['2007-03/2007']  # March 2007 to the end of 2007
sample.xts['/'] # the whole data set
sample.xts['/2007'] # the beginning of the data through 2007
sample.xts['2007-01-03'] # just the 3rd of January 2007
```

---

xtsAPI *xts C API Documentation*

Description

This help file is to help in development of xts, as well as provide some clarity and insight into its purpose and implementation.

By Jeffrey A. Ryan, Dirk Eddelbuettel, and Joshua M. Ulrich Last modified: 2018-05-02 Version: 0.10-3 and above

At present the **xts** API has publicly available interfaces to the following functions (as defined in `xtsAPI.h`):

```
Callable from other R packages:
  SEXP xtsIsOrdered(SEXP x, SEXP increasing, SEXP strictly)
  SEXP xtsNaCheck(SEXP x, SEXP check)
  SEXP xtsTry(SEXP x)
  SEXP xtsRbind(SEXP x, SEXP y, SEXP dup)
  SEXP xtsCoredata(SEXP x)
  SEXP xtsLag(SEXP x, SEXP k, SEXP pad)

Internal use functions:
  SEXP isXts(SEXP x)
  void copy_xtsAttributes(SEXP x, SEXP y)
  void copy_xtsCoreAttributes(SEXP x, SEXP y)

Internal use macros:
  xts_ATTRIB(x)
  xts_COREATTRIB(x)
  GET_xtsIndex(x)
  SET_xtsIndex(x,value)
  GET_xtsIndexFormat(x)
  SET_xtsIndexFormat(x,value)
  GET_xtsCLASS(x)
  SET_xtsCLASS(x,value)

Internal use SYMBOLS:
  xts_IndexSymbol
  xts_ClassSymbol
  xts_IndexFormatSymbol
```

```
Callable from R:
  SEXP mergeXts(SEXP args)
  SEXP rbindXts(SEXP args)
  SEXP tryXts(SEXP x)
```

Author(s)

Jeffrey A. Ryan

Examples

```
## Not run:
# some example code to look at

file.show(system.file('api_example/README', package="xts"))
file.show(system.file('api_example/src/checkOrder.c', package="xts"))

## End(Not run)
```

---

xtsAttributes *Extract and Replace xts Attributes*

Description

Extract and replace non-core `xts` attributes.

Usage

```
xtsAttributes(x, user=NULL)

xtsAttributes(x) <- value
```

Arguments

`x`	an xts object
`user`	logical; should user-defined attributes be returned? The default of `NULL` returns all `xts` attributes.
`value`	a list of new name=value attributes

Details

Since `xts` objects are S3 objects with special attributes, a method is necessary to properly assign and view the user-added attributes.

A call to `attributes` from the **base** package will return all attributes, including those specific to the `xts` class.

Value

A named list of user settable attributes.

Author(s)

Jeffrey A. Ryan

See Also

attributes

Examples

```
x <- xts(matrix(1:(9*6),nc=6),
         order.by=as.Date(13000,origin="1970-01-01")+1:9,
         a1='my attribute')

xtsAttributes(x)
xtsAttributes(x) <- list(a2=2020)

xtsAttributes(x)
xtsAttributes(x) <- list(a1=NULL)
xtsAttributes(x)
```

---

xtsInternals *Internal Documentation*

Description

This help file is to help in development of xts, as well as provide some clarity and insight into its purpose and implementation.

Last modified: 2008-08-06 by Jeffrey A. Ryan Version: 0.5-0 and above

The **xts** package xts designed as a drop-in replacement for the very popular **zoo** package. Most all functionality of zoo has been extended or carries into the xts package.

Notable changes in direction include the use of time-based indexing, at first explicitely, now implicitely.

An xts object consists of data in the form of a matrix, an index - ordered and increasing, either numeric or integer, and additional attributes for use internally, or for end-user purposes.

The current implementation enforces two major rules on the object. One is that the index must be coercible to numeric, by way of as.POSIXct. There are defined types that meet this criteria. See timeBased for details.

The second requirement is that the object cannot have rownames. The motivation from this comes in part from the work Matthew Doyle has done in his data.table class, in the package of the same name. Rownames in R must be character vectors, and as such are inefficient in both storage and conversion. By eliminating the rownames, and providing a numeric index of R internal type REAL or

INTEGER, it is possible to maintain a connection to standard R date and time classes via the POSIXct functions, while at at the same time maximizing efficiencies in data handling.

User level functions index, as well as conversion to other classes proceeds as if there were rownames. The code for index automatically converts time to numeric in both extraction and replacement functionality. This provides a level of abstraction to facilitate internal, and external package use and inter-operability.

There is also new work on providing a C-level API to some of the xts functionality to facilitate external package developers to utilize the fast utility routines such as subsetting and merges, without having to call only from R. Obviously this places far more burden on the developer to not only understand the internal xts implementation, but also to understand all of what is documented for R-internals (and much that isn't). At present the functions and macros available can be found in the 'xts.h' file in the src directory.

There is no current documentation for this API. The adventure starts here. Future documentation is planned, not implemented.

Author(s)

Jeffrey A. Ryan

[.xts *Extract Subsets of xts Objects*

Description

Details on efficient subsetting of xts objects for maximum performance and compatibility.

Usage

```
## S3 method for class 'xts'
x[i, j, drop = FALSE, which.i=FALSE, ...]
```

Arguments

x	xts object
i	the rows to extract. Numeric, timeBased or ISO-8601 style (see details)
j	the columns to extract, numeric or by name
drop	should dimension be dropped, if possible. See NOTE.
which.i	return the 'i' values used for subsetting. No subset will be performed.
...	additional arguments (unused)

Details

One of the primary motivations, and key points of differentiation of the time series class xts, is the ability to subset rows by specifying ISO-8601 compatible range strings. This allows for natural range-based time queries without requiring prior knowledge of the underlying time object used in construction.

When a raw character vector is used for the `i` subset argument, it is processed as if it was ISO-8601 compliant. This means that it is parsed from left to right, according to the following specification:

CCYYMMDD HH:MM:SS.ss+

A full description will be expanded from a left-specified truncated one.

Additionally, one may specify range-based queries by simply supplying two time descriptions seperated by a forward slash:

CCYYMMDD HH:MM:SS.ss+/CCYYMMDD HH:MM:SS.ss

The algorithm to parse the above is `.parseISO8601` from the **xts** package.

ISO-style subsetting, given a range type query, makes use of a custom binary search mechanism that allows for very fast subsetting as no linear search though the index is required. ISO-style character vectors may be longer than length one, allowing for multiple non-contiguous ranges to be selected in one subsetting call.

If a character *vector* representing time is used in place of numeric values, ISO-style queries, or timeBased objects, the above parsing will be carried out on each element of the i-vector. This overhead can be very costly. If the character approach is used when no ISO range querying is needed, it is recommended to wrap the 'i' character vector with the `I()` function call, to allow for more efficient internal processing. Alternately converting character vectors to POSIXct objects will provide the most performance efficiency.

As `xts` uses POSIXct time representations of all user-level index classes internally, the fastest timeBased subsetting will always be from POSIXct objects, regardless of the `tclass` of the original object. All non-POSIXct time classes are converted to character first to preserve consistent TZ behavior.

Value

An extraction of the original xts object. If `which.i` is TRUE, the corresponding integer 'i' values used to subset will be returned.

Note

By design, drop=FALSE in the default case. This preserves the basic underlying type of `matrix` and the `dim()` to be non-NULL. This is different from both matrix and `zoo` behavior as R uses `drop=TRUE`. Explicitly passing `drop=TRUE` may be required when performing certain matrix operations.

Author(s)

Jeffrey A. Ryan

References

ISO 8601: Date elements and interchange formats - Information interchange - Representation of dates and time https://www.iso.org

See Also

xts, .parseISO8601, .index

Examples

```
x <- xts(1:3, Sys.Date()+1:3)
xx <- cbind(x,x)

# drop=FALSE for xts, differs from zoo and matrix
z <- as.zoo(xx)
z/z[,1]

m <- as.matrix(xx)
m/m[,1]

# this will fail with non-conformable arrays (both retain dim)
tryCatch(
  xx/x[,1],
  error=function(e) print("need to set drop=TRUE")
)

# correct way
xx/xx[,1,drop=TRUE]

# or less efficiently
xx/drop(xx[,1])
# likewise
xx/coredata(xx)[,1]


x <- xts(1:1000, as.Date("2000-01-01")+1:1000)
y <- xts(1:1000, as.POSIXct(format(as.Date("2000-01-01")+1:1000)))

x.subset <- index(x)[1:20]
x[x.subset] # by original index type
system.time(x[x.subset])
x[as.character(x.subset)] # by character string. Beware!
system.time(x[as.character(x.subset)]) # slow!
system.time(x[I(as.character(x.subset))]) # wrapped with I(), faster!

x['200001'] # January 2000
x['1999/2000'] # All of 2000 (note there is no need to use the exact start)
x['1999/200001'] # January 2000

x['2000/200005'] # 2000-01 to 2000-05
x['2000/2000-04-01'] # through April 01, 2000
y['2000/2000-04-01'] # through April 01, 2000 (using POSIXct series)
```

```
### Time of day subsetting

i <- 0:60000
focal_date <- as.numeric(as.POSIXct("2018-02-01", tz = "UTC"))
x <- .xts(i, c(focal_date + i * 15), tz = "UTC", dimnames = list(NULL, "value"))

# Select all observations between 9am and 15:59:59.99999:
w1 <- x["T09/T15"] # or x["T9/T15"]
head(w1)

# timestring is of the form THH:MM:SS.ss/THH:MM:SS.ss

# Select all observations between 13:00:00 and 13:59:59.9999 in two ways:
y1 <- x["T13/T13"]
head(y1)

x[.indexhour(x) == 13]

# Select all observations between 9:30am and 30 seconds, and 4.10pm:
x["T09:30:30/T16:10"]

# It is possible to subset time of day overnight.
# e.g. This is useful for subsetting FX time series which trade 24 hours on week days

# Select all observations between 23:50 and 00:15 the following day, in the xts time zone
z <- x["T23:50/T00:14"]
z["2018-02-10 12:00/"] # check the last day


# Select all observations between 7pm and 8.30am the following day:
z2 <- x["T19:00/T08:29:59"]
head(z2); tail(z2)
```

Index