

Package ‘xts’

September 8, 2026

Type Package

Title eXtensible Time Series

Version 0.14.3

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>

Encoding UTF-8

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 2026-09-08 16:50:02 UTC

Contents

xts-package	3
.parseISO8601	3

addEventLines	5
addLegend	6
addPanel	7
addPolygon	8
addSeries	9
adj.time	10
apply.daily	11
as.environment.xts	13
as.xts.Date	14
axTicksByTime	16
c.xts	18
CLASS	19
coredata.xts	20
dimnames.xts	21
endpoints	22
first	23
firstof	25
index.xts	26
indexTZ	30
is.index.unique	32
is.timeBased	33
isOrdered	34
lag.xts	35
merge.xts	37
na.locf.xts	39
nseconds	40
period.apply	41
period.sum	43
periodicity	44
plot.xts	46
print.xts	49
sample_matrix	50
split.xts	51
tclass	52
tformat	54
timeBasedRange	55
to.period	57
try.xts	59
window.xts	61
xts	62
xts-internals	65
xtsAPI	66
xtsAttributes	67
[.xts	68

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 replicates most of the ISO standard for parsing times and time-based ranges in a universally accepted way. The best documentation is the official ISO page as well as the Wikipedia entry for ISO 8601:2004.

Usage

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

Arguments

x	A character string conforming to the ISO 8601:2004(e) rules.
start	Lower constraint on range.
end	Upper constraint of range
tz	Timezone (tzone) to use internally.

Details

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 `as.POSIXct()` 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 two element list with an entry named 'first.time' and one named 'last.time'.

Note

There is no checking done to test for a properly constructed ISO format string. This must be correctly entered by the user.

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, 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	<i>Add vertical lines to an existing xts plot</i>
---------------	---

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	An xts object of events and their associated labels. It is ensured 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	<i>Add Legend</i>
-----------	-------------------

Description

Add a legend to an existing panel.

Usage

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

Arguments

<code>legend.loc</code>	One of nine locations: bottomright, bottom, bottomleft, left, topleft, top, topright, right, or center.
<code>legend.names</code>	Character vector of names for the legend. When NULL, the column names of the current plot object are used.
<code>col</code>	Fill colors for the legend. When NULL, the colorset of the current plot object data is used.
<code>ncol</code>	Number of columns for the legend.
<code>on</code>	Panel number to draw on. A new panel will be drawn if <code>on = NA</code> . The default, <code>on = 0</code> , 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 <code>on</code> is checked for the default behavior to add to the last active panel.
<code>...</code>	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 on an existing or new panel. 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

See Also

[plot.xts\(\)](#), [addSeries\(\)](#)

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 the lower y coordinates for the polygon. The first column is interpreted as 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.
...	Any other passthrough 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 add to the 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 <code>plot()</code> .
col	Color palette to use, set by default to rational choices.
lty	Set the line type, same as in <code>par()</code> .
lwd	Set the line width, same as in <code>par()</code> .
pch	The type of plot to be drawn, same as in <code>par()</code> .
...	Any other passthrough graphical parameters.

Author(s)

Ross Bennett

adj.time	<i>Align seconds, minutes, and hours to beginning of next period.</i>
----------	---

Description

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

Usage

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

align.time(x, ...)

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

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

Arguments

x	Object containing timestamps to align.
...	Additional arguments. See details.
n	Number of seconds to adjust by.

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 with the same class as 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.daily

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	A time-series object coercible to xts.
FUN	A function to apply to each period.
...	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` previously worked 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	<i>Coerce an xts Object to an Environment by Column</i>
--------------------	---

Description

Method to automatically convert an xts object to an environment containing vectors representing each column of the original xts object. The name of each object in the resulting environment corresponds to the name of the column of the xts object.

Usage

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

Arguments

x An xts object.

Value

An environment containing `ncol(x)` vectors extracted by column from x.

Note

Environments do not preserve (or have knowledge) of column order and cannot be subset by an integer index.

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(y) <- "Y"  
xy <- cbind(x,y)  
colnames(xy)  
e <- as.environment(xy)    # currently using xts-style positive k  
ls(xy)  
ls.str(xy)
```

as.xts.Date

*Convert Objects To and From xts***Description**

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

Usage

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

## S3 method for class 'POSIXt'
as.xts(x, ...)

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

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

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

## S3 method for class 'timeDate'
as.xts(x, ...)

## S3 method for class 'timeSeries'
as.xts(
  x,
  dateFormat = "POSIXct",
  FinCenter,
```

```

    recordIDs,
    title,
    documentation,
    ...,
    .RECLASS = FALSE
)

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

as.xts(x, ...)

xtsible(x)

## S3 method for class 'yearmon'
as.xts(x, ...)

## S3 method for class 'yearqtr'
as.xts(x, ...)

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

```

Arguments

<code>x</code>	Data object to convert. See details for supported types.
<code>...</code>	Additional parameters or attributes.
<code>order.by, frequency</code>	See zoo help.
<code>dateFormat</code>	What class should the dates be converted to?
<code>.RECLASS</code>	Should the conversion be reversible via reclass() ?
<code>FinCenter, recordIDs, title, documentation</code>	See timeSeries help.

Details

A simple and reliable way to convert many different objects into a uniform format for use within R. `as.xts()` can convert objects of the following classes into an xts object: object: [timeSeries](#), [ts](#), [matrix](#), [data.frame](#), and [zoo](#). `xtsible()` safely checks whether an object can be converted to an xts object.

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

When `.RECLASS = TRUE`, the returned xts object internally preserves all relevant attribute/slot data from the input `x`. This allows for temporary conversion to xts in order to use zoo and xts compatible methods. See [reclass\(\)](#) for details.

Value

An S3 object of class xts.

Author(s)

Jeffrey A. Ryan

See Also

`xts()`, `reclass()`, `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. This function is written for internal use, and documented for those wishing to use it for customized plots.

Usage

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

Arguments

x	An object indexed by time or a vector of times/dates.
ticks.on	Time unit for tick locations.
k	Frequency of tick locations.
labels	Should a labeled vector be returned?
format.labels	Either a logical value specifying whether labels should be formatted, or a character string specifying the format to use.
ends	Should the ends be adjusted?
gt	Lower bound on number of tick locations.
lt	Upper bound on number of tick locations.

Details

The default `ticks.on = "auto"` uses heuristics to compute sensible tick locations. Use a combination of `ticks.on` and `k` to create tick locations at specific intervals. For example, `ticks.on = "days"` and `k = 7` will create tick marks every 7 days.

When `format.labels` is a character string the possible values are the same as those listed in the Details section of [strptime\(\)](#).

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)
```

`c.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. All objects must have the same number of columns and be xts objects or coercible to xts.

Usage

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

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

Arguments

`...` Objects to bind by row.
`deparse.level` Not implemented.

Details

Duplicate index values are supported. When one or more input has the same index value, the duplicated index values in the result are in the same order the objects are passed to `rbind()`. See examples.

`c()` is an alias for `rbind()` for xts objects.

See [merge.xts\(\)](#) for traditional merge operations.

Value

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

Note

`rbind()` is a `'Primitive'` function in R, which means method dispatch occurs at the C-level, and may not be consistent with normal S3 method dispatch (see [rbind\(\)](#) for details). Call `rbind.xts()` directly to avoid potential 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)

```

CLASS

Extract and Set .CLASS Attribute

Description

Extraction and replacement functions to access the 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

<code>x</code>	An xts object.
<code>value</code>	The new value to assign to the <code>'CLASS'</code> attribute.

Details

This is meant for use in conjunction with `try.xts()` and `reclass()` and is not intended for daily use. While it's possible to interactively coerce objects to other classes than originally derived from, it's likely to cause unexpected behavior. It is best to use the usual `as.xts()` and other classes' 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 formatted using <code>tformat()</code> ? Alternatively a date/time string to be passed to <code>format()</code> . See details.
...	Unused.
value	Non-core attributes to assign.

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 rownames of the result use the format specified by `tformat(x)` when `fmt = TRUE`. When `fmt` is a character string to be passed to `format()`. See [strptime\(\)](#) for valid format strings. Setting `fmt = FALSE` will return the row names by simply coercing the index class to a character string in the default manner.

`xcoredata()` is the complement to `coredata()`. It returns all of the attributes normally removed by `coredata()`. Its purpose, along with the replacement function `xcoredata<-` is primarily for developers using `xts`' [try.xts\(\)](#) and [reclass\(\)](#) functionality inside functions so the functions can take any time series class as an input and return the same time series class.

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)
```

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 two element list. See Details.

Details

For efficiency, xts objects do not have rownames (unlike zoo objects). Attempts to set rownames on an xts object will silently set them to NULL. This is done for internal compatibility reasons, as well as to provide consistency in performance regardless of object use.

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.

Note

Unlike zoo, all xts objects have dimensions. xts objects cannot be plain 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 and k.

Usage

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

Arguments

x	An xts object.
on	A character string specifying the period.
k	The number of periods each endpoint should cover.

Details

`endpoints()` returns a numeric vector that always begins with zero and ends with the number of observations in `x`.

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

Valid values for the `on` argument are: “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 number of of observations in `x`.

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

Generic functions to return the first or last elements or rows of a vector or two-dimensional data object.

Usage

```

first(x, ...)

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

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

last(x, ...)

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

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

```

Arguments

x	An object.
...	Arguments passed to other methods.
n	Number of observations to return.
keep	Should removed values be kept as an attribute on the result?

Details

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

Quickly and easily extract the first or last *n* observations of an object. When *n* is a number, these functions are similar to `head()` and `tail()`, but only return the *first* or *last* observation by default.

n can be a character string if *x* is an xts object or coerceable to xts. It must be of the form ‘*n* period’, where ‘*n*’ is a numeric value (1 if not provided) describing the number of periods to return. Valid periods are: secs, seconds, mins, minutes, hours, days, weeks, months, quarters, and years.

The ‘period’ portion can be any frequency greater than or equal to the frequency of the object’s time index. For example, `first(x, "2 months")` will return the first 2 months of data even if *x* is hourly frequency. Attempts to set ‘period’ to a frequency less than the object’s frequency will throw an error.

n may be positive or negative, whether it’s a number or character string. When *n* is positive, the functions return the obvious result. For example, `first(x, "1 month")` returns the first month’s data. When *n* is negative, all data *except* first month’s is returned.

Requesting more data than is in *x* will throw a warning and simply return *x*.

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

Easily create 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 = "")

lastof(
  year = 1970,
  month = 12,
  day = 31,
  hour = 23,
  min = 59,
  sec = 59,
  subsec = 0.99999,
  tz = ""
)
```

Arguments

year, month, day	Numeric values to specify a day.
hour, min, sec	Numeric vaues to specify time within a day.
tz	Timezone used for conversion.
subsec	Number of sub-seconds.

Details

This is a wrapper to [ISOdatetime\(\)](#) with defaults corresponding to the first or last possible time in a given period.

Value

An POSIXct object.

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

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

## S3 method for class 'xts'
time(x, ...)

.index(x, ...)
```

```
.index(x) <- value  
  
.indexsec(x)  
  
.indexmin(x)  
  
.indexhour(x)  
  
.indexmday(x)  
  
.indexmon(x)  
  
.indexyear(x)  
  
.indexwday(x)  
  
.indexbday(x)  
  
.indexyday(x)  
  
.indexisdst(x)  
  
.indexDate(x)  
  
.indexday(x)  
  
.indexweek(x)  
  
.indexyweek(x)  
  
convertIndex(x, value)
```

Arguments

x	An xts object.
...	Arguments passed to other methods.
value	A new time index value.

Details

An xts object's index is stored internally as the number of seconds since UNIX 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 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 internal numeric 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 below extract time components from the internal time index. They return values like the values of [POSIXlt](#) components.

```
.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)
.indexyday 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 `tzzone()`, `tclass()`, and `tformat()` versions. The previous versions throw a warning that they're deprecated, but they will continue to work. They will never be removed or throw an error. Ever.

The new 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.

You can set `options(xts.warn.index.missing.tzone = TRUE)` and `options(xts.warn.index.missing.tclass = TRUE)` to identify xts objects that do not have a 'tzone' or 'tclass' attribute on the index, even if there is a 'tzone' or 'tclass' attribute on the xts object itself. The warnings will be thrown when the object is printed. Use `x <- as.xts(x)` to update these objects to the new structure.

Version 0.9.8: The index timezone is now set to "UTC" for time classes that do not have any intra-day 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: `tzzone()` and `tzzone<=`, `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()` documents how **xts** handles the index class, and `tzzone()` has more information about 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]

```

indexTZ

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

```
indexTZ(x, ...)
```

```
tzzone(x, ...)
```

```
indexTZ(x) <- value
```

```
tzzone(x) <- value
```

Arguments

x	An xts object.
...	Arguments passed to other methods.
value	A valid timezone value (see OlsonNames()).

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 tzzone<- 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 tzzone() and tzzone<-, 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).

Author(s)

Jeffrey A. Ryan

See Also

[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())
tzzone(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!
```

is.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

```
is.index.unique(x)
```

```
is.time.unique(x)
```

```
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	A value to add to force uniqueness.

drop	Should duplicates be dropped instead of adjusted by eps?
fromLast	When drop = TRUE, fromLast controls which duplicated times are dropped. When fromLast = FALSE, the earliest observation with an identical timestamp is kept and subsequent observations are 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 with unique timestamps.

Note

Incoming values must be pre-sorted, and no check is done to make sure that this is the case. ‘integer’ index value will be coerced to ‘double’ when drop = FALSE.

Author(s)

Jeffrey A. Ryan

See Also

[align.time\(\)](#)

Examples

```
ds <- options(digits=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)
```

is.timeBased

Check if Class is Time-Based

Description

Used to verify that the object is one of the known time-based classes in R. Current time-based objects supported are Date, POSIXct, chron, yearmon, yearqtr, and timeDate.

Usage

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

```
timeBased(x)
```

Arguments

`x` Object to test.

Value

A logical scalar.

Author(s)

Jeffrey A. Ryan

Examples

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

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

```
timeBased(200701)
```

isOrdered

Check If A Vector Is Ordered

Description

Check 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 (TRUE) or decreasing (FALSE) values?

`strictly` When TRUE, vectors with duplicate values are *not* considered ordered.

Details

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

Value

A logical scalar indicating whether or not x is ordered.

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)
```

lag.xts

Lags and Differences of xts Objects

Description

Methods for computing lags and differences on xts objects. This provides similar functionality as the **zoo** counterparts, but with some different defaults.

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	Number of periods to shift.
na.pad	Should NA be added so the result has the same number of observations as x?
...	Additional arguments.
lag	Period to difference over.
differences	Order of differencing.
arithmetic	Should arithmetic or geometric differencing be used?
log	Should (geometric) log differences be returned?

Details

The primary motivation for these methods was to take advantage of a faster C-level implementation. Another motivation was to make `lag()` behave using standard sign for `k`. Both [zoo's lag\(\) method](#) and `lag.default()` require a *negative* value for `k` in order to shift a series backward. So `k = 1`, shifts the series *forward* one observation. This is especially confusing because `k = 1` is the default for those functions. When `x` is an xts object, `lag(x, 1)` returns an object where the value at time 't' is the value at time 't-1' in the original object.

Another difference is that `na.pad = TRUE` by default, to better reflect the transformation visually and for functions the require positional alignment of data.

Set `options(xts.compat.zoo.lag = TRUE)` to use make `lag.xts()` consistent with `lag.zoo()` by reversing the sign of `k` and setting `na.pad = FALSE`.

Value

An xts object with 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))
```

merge.xts

Merge xts Objects

Description

Perform merge operations on xts objects by time index.

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
)

## S3 method for class 'xts'
cbind(..., all = TRUE, fill = NA, suffixes = 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	Suffix to be added to merged column names.
join	Type of database join. One of 'outer', 'inner', 'left', or 'right'.
retside	Which side of the merged object should be returned (2-case only)?
retclass	Either a logical value indicating whether the result should have a 'class' attribute, or the name of the desired class for the result.
tzone	Time zone to use for the merged result.
drop	Not currently used.
check.names	Use make.names() to ensure column names are valid R object names?

Details

This xts method is compatible with [zoo's merge\(\) method](#) but implemented almost entirely in C-level code for efficiency.

The function can perform all common database join operations along the time index by setting 'join' to one of the values below. Note that 'left' and 'right' are only implemented for two objects.

- outer: full outer (all rows in all objects)
- inner: only rows with common indexes in all objects
- left: all rows in the first object, and rows from the second object that have the same index as the first object
- right: all rows in the second object, and rows from the first object that have the same index as the second object

The above join types can also be accomplished by setting 'all' to one of the values below.

- outer: all = TRUE or all = c(TRUE, TRUE)
- inner: all = FALSE or all = c(FALSE, FALSE)
- left: all = c(TRUE, FALSE)
- right: all = c(FALSE, TRUE)

The result will have the timezone of the leftmost argument if available. Use the 'tzone' argument to override the default behavior.

When retclass = 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, but unused by xts. When retclass = FALSE the object will be stripped of its class attribute. This is for internal use.

See the examples in order to join using an 'all' argument that is the same arguments to join, like you can do with `merge.zoo()`.

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://learn.microsoft.com/en-us/archive/blogs/craigfr/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 indicating whether leading/trailing NA should be removed. The default is FALSE unlike the zoo method.
fromLast	Logical indicating whether observations should be carried backward rather than forward. Default is FALSE.
maxgap	Consecutive runs of observations more than 'maxgap' will remain NA. See na.locf() for details.
...	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 is to leave trailing or leading NA's in place. This is different than **zoo** behavior.

Value

An object where each NA in `object` is replaced by the most recent non-NA prior to it. See [na.locf\(\)](#) for details.

Author(s)

Jeffrey A. Ryan

See Also

[na.locf\(\)](#)

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)
```

nseconds

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 result is the number of endpoints found.

As a compromise between simplicity and accuracy, the results will always round up to the nearest complete period. Subtract 1 from the result to get the completed periods.

For finer grain detail one should call the higher frequency functions.

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

Value

The number of respective periods in `x`.

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

<code>x</code>	The data that FUN will be applied to.
<code>INDEX</code>	A numeric vector of index breakpoint locations. The vector should begin with 0 and end with <code>nrow(x)</code> .
<code>FUN</code>	A function to apply to each interval in <code>x</code> .
<code>...</code>	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` previously worked 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.sum

Optimized Calculations By Period

Description

Calculate a sum, product, minimum, or maximum for each non-overlapping period specified by INDEX.

Usage

```

period.sum(x, INDEX)

period.prod(x, INDEX)

period.max(x, INDEX)

period.min(x, INDEX)

```

Arguments

x	A univariate data object.
INDEX	A numeric vector of endpoints for each period.

Details

These functions are similar to calling `period.apply()` with the same endpoints and function. There may be slight differences in the results due to numerical accuracy.

For xts-coercible objects, an appropriate INDEX can be created by a call to `endpoints()`.

Value

An xts or zoo object containing the sum, product, minimum, or maximum for each endpoint in INDEX.

Author(s)

Jeffrey A. Ryan

See Also

`endpoints()`, `period.apply()`

Examples

```
x <- c(1, 1, 4, 2, 2, 6, 7, 8, -1, 20)
i <- c(0, 3, 5, 8, 10)
```

```
period.sum(x, i)
period.prod(x, i)
period.min(x, i)
period.max(x, i)
```

```
data(sample_matrix)
y <- sample_matrix[, 1]
ep <- endpoints(sample_matrix)
```

```
period.sum(y, ep)
period.sum(as.xts(y), ep)
```

```
period.prod(y, ep)
period.prod(as.xts(y), ep)
```

```
period.min(y, ep)
period.min(as.xts(y), ep)
```

```
period.max(y, ep)
period.max(as.xts(y), ep)
```

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

<code>x</code>	A time-series-like object.
<code>...</code>	Unused.

Details

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

This calculates the median time difference 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 a `'Date'` or `'POSIXct'` class, or or coercible to one of them.

The `'scale'` component of the result 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.

Value

A `'periodicity'` object with the following elements:

- the `difftime` object,
- frequency: the median time difference between observations
- start: the first observation
- end: the last observation
- units: one of secs, mins, hours, or days
- scale: one of seconds, minute, hourly, daily, weekly, monthly, quarterly, or yearly
- label: one of second, minute, hour, day, week, month, quarter, year

Possible scale values are: `'minute'`, `'hourly'`, `'daily'`, `'weekly'`, `'monthly'`, `'quarterly'`, and `'yearly'`.

Note

This function only attempts to be a *good estimate* for the underlying periodicity. If the series is too short, or has highly irregular periodicity, the return values will not be accurate. 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

<code>x</code>	A xts object.
<code>y</code>	Not used, always NULL.
<code>...</code>	Any passthrough arguments for <code>lines()</code> and <code>points()</code> .
<code>subset</code>	An ISO8601-style subset string.
<code>panels</code>	Character vector of expressions to plot as panels.
<code>multi.panel</code>	Either TRUE, FALSE, or an integer less than or equal to the number of columns in the data set. When TRUE, each column of the data is plotted in a separate panel. When an integer 'n', the data will be plotted in groups of 'n' columns per panel and each group will be plotted in a separate panel.
<code>col</code>	Color palette to use.
<code>up.col</code>	Color for positive bars when <code>type = "h"</code> .
<code>dn.col</code>	Color for negative bars when <code>type = "h"</code> .
<code>bg</code>	Background color of plotting area, same as in <code>par()</code> .
<code>type</code>	The type of plot to be drawn, same as in <code>plot()</code> .
<code>lty</code>	Set the line type, same as in <code>par()</code> .
<code>lwd</code>	Set the line width, same as in <code>par()</code> .
<code>lend</code>	Set the line end style, same as in <code>par()</code> .
<code>main</code>	Main plot title.
<code>main.timespan</code>	Should the timespan of the series be shown in the top right corner of the plot?
<code>observation.based</code>	When TRUE, all the observations are equally spaced along the x-axis. When FALSE (the default) the observations on the x-axis are spaced based on the time index of the data.
<code>log</code>	Should the y-axis be in log scale? Default FALSE.
<code>ylim</code>	The range of the y axis.

<code>yaxis.same</code>	Should 'ylim' be the same for every panel? Default TRUE.
<code>yaxis.left</code>	Add y-axis labels to the left side of the plot?
<code>yaxis.right</code>	Add y-axis labels to the right side of the plot?
<code>yaxis.ticks</code>	Desired number of y-axis grid lines. The actual number of grid lines is determined by the <code>n</code> argument to pretty() .
<code>major.ticks</code>	Period specifying locations for major tick marks and labels on the x-axis. See Details for possible values.
<code>minor.ticks</code>	Period specifying locations for minor tick marks on the x-axis. When NULL, minor ticks are not drawn. See details for possible values.
<code>grid.ticks.on</code>	Period specifying locations for vertical grid lines. See details for possible values.
<code>grid.ticks.lwd</code>	Line width of the grid.
<code>grid.ticks.lty</code>	Line type of the grid.
<code>grid.col</code>	Color of the grid.
<code>labels.col</code>	Color of the axis labels.
<code>format.labels</code>	Label format to draw lower frequency x-axis ticks and labels passed to axTicksByTime()
<code>grid2</code>	Color for secondary x-axis grid.
<code>legend.loc</code>	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.
<code>extend.xaxis</code>	When TRUE, extend the x-axis before and/or after the plot's existing time index range, so all of the time index values of the new series are included in the plot. Default FALSE.
<code>on</code>	Panel number to draw on. A new panel will be drawn if <code>on = NA</code> . The default, <code>on = 0</code> , 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 <code>on</code> is checked for the default behavior to add to the last active panel.
<code>pch</code>	the plotting character to use, same as in 'par'

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 **quantmod** written 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

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

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)
```

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	<i>Divide into Groups by Time</i>
-----------	-----------------------------------

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 <code>split.xts()</code> .
k	Number of periods to aggregate into each split. See details.
...	Further arguments passed to other methods.

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.

The inputs are passed to [split.zoo\(\)](#) when f is not a character vector.

Value

A list of xts objects.

Note

[aggregate.zoo\(\)](#) is 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, ...)
```

Default S3 method:

```
tclass(x, ...)
```

S3 method for class 'xts'

```
tclass(x, ...)
```

tclass(x) <- value

Default S3 replacement method:

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

indexClass(x)

indexClass(x) <- value

S3 replacement method for class 'xts'

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

Arguments

x	An xts object.
...	Arguments passed to other methods.
value	The new index class (see Details for valid values).

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` can *potentially change* the values of the internal index. For example, changing the 'tclass' from `POSIXct` to `Date` will truncate the `POSIXct` value and convert the timezone to UTC (since the `Date` class doesn't have a timezone). 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, and [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 *changes* the internal index values
head(y)           # the index has times
head(.index(y))
```

```
tclass(y) <- "Date"
head(y)          # the index prints as a Date
head(.index(y))  # the internal index is truncated
```

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

indexFormat(x)

indexFormat(x) <- value
```

Arguments

x	An xts object.
...	Arguments passed to other methods.
value	New index format string (see strptime() details for valid values).

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.).

The 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)
```

timeBasedRange

*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 resulting series can be of any class supported by xts, including POSIXct, Date, chron, timeDate, yearmon, and yearqtr.

Usage

```
timeBasedRange(x, ...)

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

Arguments

<code>x</code>	An ISO-8601 time-date range string.
<code>...</code>	Unused.
<code>retclass</code>	The return class desired.
<code>length.out</code>	Passed to <code>seq()</code> internally.

Details

`timeBasedRange()` creates a one or two element numeric vector representing the start and end number of seconds since epoch (1970-01-01). For internal use.

`timeBasedSeq()` creates sequences of time-based observations using strings formatted according to the ISO-8601 specification. The general format is *from/to/by* or *from::to::by*, where *to* and *by* are optional when the 'length.out' argument is specified.

The *from* and *to* elements of the string must be left-specified with respect to the standard *CCYYM-MDD HHMMSS* form. All dates/times specified will be set to either the earliest point (from) or the latest (to), to the given level of specificity. For example, '1999' in the *from* field would set the start to the beginning of 1999. '1999' in the *to* field would set the end to the end of 1999.

The amount of resolution in the result is determined by the resolution of the *from* and *to* component, unless the optional *by* component is specified.

For example, `timeBasedSeq("1999/2008")` returns a vector of Dates for January 1st of each year. `timeBasedSeq("199501/1996")` returns a yearmon vector of 24 months in 1995 and 1996. And `timeBasedSeq("19950101/1996")` creates a Date vector for all the days in those two years.

The optional *by* field (the third delimited element to the string), will use the resolution heuristic described above and will use the specified *by* resolution. The possible values for *by* are: 'Y' (years), 'm' (months), 'd' (days), 'H' (hours), 'M' (minutes), 'S' (seconds). Sub-second resolutions are not supported.

Value

`timeBasedSeq()` returns a vector of time-based observations. `timeBasedRange()` returns a one or two element numeric vector representing the start and end number of seconds since epoch (1970-01-01).

When `retclass = NULL`, the result of `timeBasedSeq()` is a named list containing elements "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.

Usage

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

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, ...)
```

Arguments

x A univariate or OHLC type time-series object.

<code>period</code>	Period to convert to. See details.
<code>k</code>	Number of sub periods to aggregate on (only for minutes and seconds).
<code>indexAt</code>	Convert final index to new class or date. See details.
<code>name</code>	Override column names?
<code>OHLC</code>	Should an OHLC object be returned? (only OHLC = TRUE currently supported)
<code>...</code>	Additional arguments.
<code>drop.time</code>	Remove time component of POSIX datestamp (if any)?

Details

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. Aggregate volume will also be calculated if applicable.

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 use the **zoo** package's [yearmon](#) and [yearqtr](#) classes, 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 = TRUE` (the default) will convert a series that includes a time component into one with just a date index, since the time component is often of little value in lower frequency series.

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 removed before aggregation, though replaced before returning the result. This inevitably leads to many additional NA values in the result. Consider using an `xts` object or converting to `xts` using `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))
```

try.xts

Convert Objects to xts and Back to Original Class

Description

Functions to convert objects of arbitrary classes to `xts` and then back to the original class, without losing any attributes of the original class.

Usage

```
try.xts(x, ..., error = TRUE)

reclass(x, match.to, error = FALSE, ...)

Reclass(x)
```

Arguments

<code>x</code>	Data object to convert. See details for supported types.
<code>...</code>	Additional parameters or attributes.
<code>error</code>	Error handling option. See Details.
<code>match.to</code>	An <code>xts</code> object whose attributes will be copied to the result.

Details

A simple and reliable way to convert many different objects into a uniform format for use within R. `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 seamless end-user experience, regardless of the end-user's choice of data-classes.

`try.xts()` calls `as.xts()` internally. See [as.xts\(\)](#) for available xts methods and arguments for each coercible class. Since it calls `as.xts()`, you can add custom attributes as `name = value` pairs in the same way. But these custom attributes will not be copied back to the original object when `reclass()` is called.

The `error` argument can be a logical value indicating whether an error should be thrown (or fail silently), a character string allowing for custom error messages, or a function of the form `f(x, ...)` that will be called if the conversion fails.

`reclass()` converts an object created by `try.xts()` back to its original class with all the original attributes intact (unless they were changed after the object was converted to xts). The `match.to` argument allows you copy the index attributes (`tcclass`, `tformat`, and `tzone`) and `xtsAttributes()` from another xts object to the result. `match.to` must be an xts object with an index value for every observation in `x`.

`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 in R are not designed to return objects matching the original object's class. It attempts to handle conversion and reversion transparently but it requires the original object must be coercible to xts, the result of the function must have the same number of rows as the input, and the object to be converted/reclassed must be the first argument to the function being wrapped. Note that this function hasn't been tested for robustness.

See the accompanying vignette for more details on the above usage.

Value

`try.xts()` returns an xts object when conversion is successful. The `error` argument controls the function's behavior when conversion fails.

`Reclass()` and `reclass()` return the object as its original class, as specified by the 'CLASS' attribute.

Author(s)

Jeffrey A. Ryan

See Also

[as.xts\(\)](#)

Examples

```
a <- 1:10

# fails silently, the result is still an integer vector
try.xts(a, error = FALSE)
```

```
# control the result with a function
try.xts(a, error = function(x, ...) { "I'm afraid I can't do that." })

z <- zoo(1:10, timeBasedSeq("2020-01-01/2020-01-10"))
x <- try.xts(z) # zoo to xts
str(x)
str(reclass(x)) # reclass back to zoo
```

window.xts

*Extract Time Windows from xts Objects***Description**

Method for extracting time windows from xts objects.

Usage

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

Arguments

<code>x</code>	An xts object.
<code>index.</code>	A user defined time index (default <code>.index(x)</code>).
<code>start</code>	A start time coercible to POSIXct.
<code>end</code>	An end time coercible to POSIXct.
<code>...</code>	Unused.

Details

The xts `window()` method provides an efficient way to subset an xts object between a start and end date using a binary search algorithm. Specifically, it converts `start` and `end` to POSIXct and then does a binary search of the index to quickly return a subset of `x` between `start` and `end`.

Both `start` and `end` may be any class that is convertible to POSIXct, such as a character string in the format 'yyyy-mm-dd'. When `start = NULL` the returned subset will begin at the first value of `index..` When `end = NULL` the returned subset will end with the last value of `index..` Otherwise the subset will contain all timestamps where `index.` is between `start` and `end`, inclusive.

When `index.` is specified, `findInterval()` is used to quickly retrieve large sets of sorted timestamps. For the best performance, `index.` must be a *sorted* POSIXct vector or a numeric vector of seconds since the epoch. `index.` is typically a subset of the timestamps in `x`.

Value

The subset of `x` that matches the time window.

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.

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

<code>x</code>	An object containing the underlying data.
<code>order.by</code>	A corresponding vector of dates/times of a known time-based class. See Details.
<code>frequency</code>	Numeric value indicating the frequency of <code>order.by</code> . See details.
<code>unique</code>	Can the index only include unique timestamps? Ignored when <code>check = FALSE</code> .
<code>tzzone</code>	Time zone of the index (ignored for indices without a time component, e.g. <code>Date</code> , <code>yearmon</code> , <code>yearqtr</code>). See tzzone() .
<code>...</code>	Additional attributes to be added. See details.
<code>index</code>	A corresponding <i>numeric</i> vector specified as seconds since the UNIX epoch (1970-01-01 00:00:00.000).
<code>tclass</code>	Time class to use for the index. See tclass() .
<code>check</code>	Must the index be ordered? The index cannot contain duplicates when <code>check = TRUE</code> and <code>unique = TRUE</code> .

Details

`xts()` is used to create an `xts` object from raw data inputs. The `xts` class inherits from and extends the `zoo` class, which means most `zoo` functions can be used on `xts` objects.

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. These attributes may be assigned and extracted via `xtsAttributes()` and `xtsAttributes<-`, respectively.

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, subsetting (`[]`), `merge()`, `cbind()`, `rbind()`, `c()`, math and logical operations, `lag()`, `diff()`, `coredata()`, `head()`, and `tail()`. There are also 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 separated by a double colon `"::"` or `"/"`. Each side of the range may be left blank, which would then default to the start 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 requires 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.

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

```

xts-internals

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.

Details

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 explicitly, now implicitly.

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 must be character vectors, and as such are inefficient in both storage and conversion. By eliminating the rownames, and providing a numeric index of internal type REAL or INTEGER, it is possible to maintain a connection to standard date and time classes via the POSIXct functions, while 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 `.`. 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

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.

Details

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	Should user-defined attributes be returned? The default of NULL returns all xts attributes.
value	A list of new name = value attributes.

Details

This function allows users to assign custom attributes to the xts objects, without altering core xts attributes (i.e. tclass, tzone, and tformat).

`attributes()` returns all attributes, including core attributes of the xts class.

Value

A named list of user-defined 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)
```

[.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

<code>x</code>	An xts object.
<code>i</code>	The rows to extract. Can be a numeric vector, time-based vector, or an ISO-8601 style range string (see details).
<code>j</code>	The columns to extract, either a numeric vector of column locations or a character vector of column names.
<code>drop</code>	Should dimension be dropped, if possible? See notes section.
<code>which.i</code>	Logical value that determines whether a subset xts object is returned (the default), or the locations of the matching rows (when <code>which.i = TRUE</code>).
<code>...</code>	Additional arguments (currently unused).

Details

One of the primary motivations and key points of differentiation of 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 class used for the time index.

When `i` is a character string, it is processed as an ISO-8601 formatted datetime or time range using `.parseISO8601()`. A single datetime is parsed from left to right, according to the following specification:

CCYYMMDD HH:MM:SS.ss+

A time range can be specified by two datetimes separated by a forward slash or double-colon. For example:

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

The ISO8601 time range subsetting uses a custom binary search algorithm to efficiently find the beginning and end of the time range. `i` can also be a vector of ISO8601 time ranges, which enables subsetting by multiple non-contiguous time ranges in one subset call.

The above parsing, both for single datetimes and time ranges, will be done on each element when `i` is a character *vector*. This is very inefficient, especially for long vectors. In this case, it's recommended to use `I(i)` so the xts subset function can process the vector more efficiently. Another alternative is to convert `i` to `POSIXct` before passing it to the subset function. See the examples for an illustration of using `I(i)`.

The xts index is stored as `POSIXct` internally, regardless of the value of its `tcclass` attribute. So the fastest time-based subsetting is always when `i` is a `POSIXct` vector.

Value

An xts object containing the subset of `x`. When `which.i = TRUE`, the corresponding integer locations of the matching rows is returned.

Note

By design, xts objects always have two dimensions. They cannot be vectors like zoo objects. Therefore `drop = FALSE` by default in order to preserve the xts object's dimensions. This is different from both matrix and zoo, which use `drop = TRUE` by default. Explicitly setting `drop = TRUE` may be needed 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

- * **chron**
 - adj.time, 10
 - lag.xts, 35
- * **datasets**
 - sample_matrix, 50
- * **manip**
 - adj.time, 10
 - as.environment.xts, 13
 - lag.xts, 35
 - merge.xts, 37
- * **misc**
 - adj.time, 10
 - dimnames.xts, 21
 - isOrdered, 34
 - na.locf.xts, 39
- * **package**
 - xts-package, 3
- * **print**
 - print.xts, 49
- * **ts**
 - adj.time, 10
 - index.xts, 26
 - indexTZ, 30
 - is.index.unique, 32
 - tclass, 52
 - tformat, 54
 - window.xts, 61
- * **utilities**
 - .parseISO8601, 3
 - [.xts, 68
 - apply.daily, 11
 - as.xts.Date, 14
 - axTicksByTime, 16
 - c.xts, 18
 - CLASS, 19
 - coredata.xts, 20
 - endpoints, 22
 - first, 23
 - firstof, 25
 - index.xts, 26
 - indexTZ, 30
 - is.timeBased, 33
 - merge.xts, 37
 - nseconds, 40
 - period.apply, 41
 - period.sum, 43
 - periodicity, 44
 - split.xts, 51
 - tclass, 52
 - tformat, 54
 - timeBasedRange, 55
 - to.period, 57
 - try.xts, 59
 - xts, 62
 - xts-internals, 65
 - xtsAPI, 66
 - xtsAttributes, 67
 - .index(index.xts), 26
 - .index(), 70
 - .index<- (index.xts), 26
 - .indexDate(index.xts), 26
 - .indexbday(index.xts), 26
 - .indexday(index.xts), 26
 - .indexhour(index.xts), 26
 - .indexisdst(index.xts), 26
 - .indexmday(index.xts), 26
 - .indexmin(index.xts), 26
 - .indexmon(index.xts), 26
 - .indexsec(index.xts), 26
 - .indexwday(index.xts), 26
 - .indexweek(index.xts), 26
 - .indexyday(index.xts), 26
 - .indexyear(index.xts), 26
 - .indexyweek(index.xts), 26
 - .parseISO8601, 3
 - .parseISO8601(), 69, 70
 - .subset.xts([.xts), 68
 - .subset_xts([.xts), 68

- .xts(xts), 62
- [.xts, 68
- addEventLines, 5
- addLegend, 6
- addPanel, 7
- addPanel(), 49
- addPolygon, 8
- addSeries, 9
- addSeries(), 7, 49
- adj.time, 10
- aggregate.zoo(), 51, 52
- align.time(adj.time), 10
- align.time(), 33
- apply.daily, 11
- apply.monthly(apply.daily), 11
- apply.monthly(), 42
- apply.quarterly(apply.daily), 11
- apply.weekly(apply.daily), 11
- apply.yearly(apply.daily), 11
- as.environment.xts, 13
- as.xts(as.xts.Date), 14
- as.xts(), 3, 20, 60, 64
- as.xts.Date, 14
- attributes(), 68
- axTicks(), 16
- axTicksByTime, 16
- axTicksByTime(), 48
- c.xts, 18
- cbind.xts(merge.xts), 37
- chart_Series(), 48
- chron(), 53
- CLASS, 19
- CLASS<- (CLASS), 19
- convertIndex(index.xts), 26
- coredata(), 21, 50
- coredata.xts, 20
- data.frame, 15
- Date(), 53
- diff.xts(lag.xts), 35
- difftime(), 45
- dimnames.xts, 21
- dimnames<- .xts(dimnames.xts), 21
- endpoints, 22
- endpoints(), 12, 17, 41, 42, 44, 51, 52, 58
- findInterval(), 61, 62
- first, 23
- firstof, 25
- head(), 24
- index(), 31, 53, 55, 64
- index.xts, 26
- index<- .xts(index.xts), 26
- indexClass(tclass), 52
- indexClass<- (tclass), 52
- indexFormat(tformat), 54
- indexFormat<- (tformat), 54
- indexTZ, 30
- indexTZ<- (indexTZ), 30
- is.index.unique, 32
- is.time.unique(is.index.unique), 32
- is.timeBased, 33
- is.unsorted(), 35
- is.xts(xts), 62
- ISO8601(.parseISO8601), 3
- ISOdatetime(), 26
- isOrdered, 34
- lag.default(), 36
- lag.xts, 35
- last(first), 23
- lastof(firstof), 25
- legend(), 6
- lines.xts(plot.xts), 46
- make.index.unique(is.index.unique), 32
- make.names(), 37
- make.time.unique(is.index.unique), 32
- matrix, 15
- merge.xts, 37
- merge.xts(), 18
- na.locf(), 39, 40
- na.locf.xts, 39
- ndays(nseconds), 40
- nhours(nseconds), 40
- nminutes(nseconds), 40
- nmonths(nseconds), 40
- nquarters(nseconds), 40
- nseconds, 40
- nweeks(nseconds), 40
- nyears(nseconds), 40
- OlsonNames(), 31

par(), [5](#), [7](#), [8](#), [10](#), [47](#)
 parseISO8601 (.parseISO8601), [3](#)
 period.apply, [41](#)
 period.apply(), [12](#), [44](#)
 period.max (period.sum), [43](#)
 period.min (period.sum), [43](#)
 period.prod (period.sum), [43](#)
 period.sum, [43](#)
 periodicity, [44](#)
 plot(), [7](#), [10](#), [47](#)
 plot.xts, [46](#)
 plot.xts(), [7](#)
 points.xts (plot.xts), [46](#)
 POSIXct(), [31](#), [53](#)
 POSIXlt, [28](#)
 pretty(), [48](#)
 print.xts, [49](#)
 print.xts(), [15](#)

 rbind(), [18](#)
 rbind.xts (c.xts), [18](#)
 Reclass (try.xts), [59](#)
 reclass (try.xts), [59](#)
 reclass(), [3](#), [15](#), [16](#), [20](#), [64](#)

 sample_matrix, [50](#)
 shift.time (adj.time), [10](#)
 split.xts, [51](#)
 split.zoo(), [51](#), [52](#)
 strptime(), [17](#), [20](#), [54](#)
 subset.xts ([.xts), [68](#)
 subset.xts(), [62](#)

 tail(), [24](#)
 tclass, [52](#), [60](#)
 tclass(), [27](#), [29](#), [31](#), [54](#), [55](#), [63](#), [64](#)
 tclass<- (tclass), [52](#)
 text(), [5](#)
 tformat, [54](#), [60](#)
 tformat(), [29](#), [31](#), [53](#), [64](#)
 tformat<- (tformat), [54](#)
 time.xts (index.xts), [26](#)
 time<- .xts (index.xts), [26](#)
 timeBased (is.timeBased), [33](#)
 timeBased(), [56](#)
 timeBasedRange, [55](#)
 timeBasedSeq (timeBasedRange), [55](#)
 timeDate(), [53](#)
 timeSeries, [15](#)

 to.daily (to.period), [57](#)
 to.hourly (to.period), [57](#)
 to.minutes (to.period), [57](#)
 to.minutes10 (to.period), [57](#)
 to.minutes15 (to.period), [57](#)
 to.minutes3 (to.period), [57](#)
 to.minutes30 (to.period), [57](#)
 to.minutes5 (to.period), [57](#)
 to.monthly (to.period), [57](#)
 to.monthly(), [12](#)
 to.period, [57](#)
 to.period(), [11](#)
 to.quarterly (to.period), [57](#)
 to.weekly (to.period), [57](#)
 to.yearly (to.period), [57](#)
 to_period (to.period), [57](#)
 try.xts, [59](#)
 try.xts(), [20](#)
 ts, [15](#)
 tzone, [60](#)
 tzone (indexTZ), [30](#)
 tzone(), [27](#), [29](#), [53](#), [55](#), [63](#), [64](#)
 tzone<- (indexTZ), [30](#)

 use.reclass (try.xts), [59](#)
 use.xts (try.xts), [59](#)

 window.xts, [61](#)

 xcoredata (coredata.xts), [20](#)
 xcoredata<- (coredata.xts), [20](#)
 xts, [62](#)
 xts(), [3](#), [16](#), [22](#), [56](#), [62](#), [70](#)
 xts-internals, [65](#)
 xts-package, [3](#)
 xtsAPI, [66](#)
 xtsAttributes, [67](#)
 xtsAttributes(), [15](#), [21](#), [60](#), [64](#)
 xtsAttributes<- (xtsAttributes), [67](#)
 xtsible (as.xts.Date), [14](#)

 yearmon, [58](#)
 yearmon(), [53](#)
 yearqtr, [58](#)
 yearqtr(), [53](#)

 zoo, [15](#)
 zoo's lag() method, [36](#)
 zoo's merge() method, [38](#)
 zoo(), [3](#), [16](#)