

Package ‘xts’

November 3, 2009

Type Package

Title Extensible Time Series

Version 0.6-9

Date 2009-10-14

Author Jeffrey A. Ryan, Josh M. Ulrich

Depends zoo

Suggests timeSeries,timeDate,tseries,its,chron,fts,tis

LazyLoad yes

Maintainer Jeffrey A. Ryan <jeff.a.ryan@gmail.com>

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-3

URL <http://r-forge.r-project.org/projects/xts/>

Repository CRAN

Date/Publication 2009-11-03 11:27:37

R topics documented:

xts-package	2
.parseISO8601	3
align.time	4
apply.monthly	5
as.xts	6
as.xts.methods	8
axTicksByTime	10
CLASS	11

coredata.xts	12
diff.xts	13
dimnames.xts	14
endpoints	16
first	17
firstof	19
indexClass	20
indexTZ	21
isOrdered	22
merge.xts	23
ndays	25
period.apply	26
period.max	27
period.min	28
period.prod	29
period.sum	30
periodicity	31
plot.xts	32
rbind.xts	34
sample_matrix	35
split.xts	36
timeBased	37
timeBasedSeq	38
to.period	39
xts	42
xtsAPI	44
xtsAttributes	45
xtsInternals	46
[.xts	47

Index **50**

xts-package *xts: extensible time-series*

Description

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

Details

Package: xts
 Type: Package
 Version: 0.6-5
 Date: 2009-05-05
 License: GPL-3

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 Josh M. Ulrich

Maintainer: Jeffrey A. Ryan <jeff.a.ryan@gmail.com>

See Also

`xts as.xts reclass zoo`

.parseISO8601	<i>Internal ISO 8601:2004(e) Time Parser</i>
---------------	--

Description

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

Usage

```
.parseISO8601(x, start, end)
```

Arguments

- x a character string conforming to ISO 8601:2004(e) rules
- start lower constraint on range
- end upper constraint of range

Details

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

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

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

Value

A list of length two, with an entry named 'first.time' and one names '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, lest bad things may happen.

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

Author(s)

Jeffrey A. Ryan

References

http://en.wikipedia.org/wiki/ISO_8601
http://www.iso.org/iso/support/faqs/faqs_widely_used_standards/widely_used_standards_other/date_and_time_format.htm

Examples

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

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

# May 5, 200 to Dec 31, 2001
.parseISO8601('2000-05/2001')

# May 5, 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')
```

align.time

Align seconds, minutes, and hours to beginning of next period.

Description

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

Usage

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

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

Arguments

x	object to align
n	number of seconds to align at
...	additional arguments

Details

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

Value

A new object of class(x)

Author(s)

Jeffrey A. Ryan with input from Brian Peterson

See Also

[to.period](#)

Examples

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

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

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

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

apply.monthly

Apply Function over Calendar Periods

Description

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

Usage

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

Arguments

<code>x</code>	an time-series object coercible to <code>xts</code>
<code>FUN</code>	an R function
<code>...</code>	additional arguments to <code>FUN</code>

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.

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

as.xts

Convert Object To And From Class xts

Description

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

Usage

```

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

Reclass(x)

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

```

Arguments

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

Details

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

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

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

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

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

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

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

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

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

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

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

Value

An S3 object of class `xts`.

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

Author(s)

Jeffrey A. Ryan

See Also

[xts,as.xts.methods](#)

as.xts.methods

Convert Object To And From Class xts

Description

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

Usage

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

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

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

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

## S3 method for class 'ts':
as.xts(x, dateFormat, ...)

## S3 method for class 'data.frame':
as.xts(x, order.by, dateFormat="POSIXct",
```

```

        frequency=NULL, ...)

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

as.fts.xts(x)

```

Arguments

<code>x</code>	data object to convert. See details for supported types
<code>dateFormat</code>	what format should the dates be converted to
<code>FinCenter</code>	see <code>timeSeries</code> help
<code>recordIDs</code>	see <code>timeSeries</code> help
<code>title</code>	see <code>timeSeries</code> help
<code>documentation</code>	see <code>timeSeries</code> help
<code>order.by</code>	see <code>zoo</code> help
<code>frequency</code>	see <code>zoo</code> help
<code>...</code>	additional parameters or attributes

Details

A simple and reliable way to convert many different objects into a uniform format for use within R. It is possible with a call to `as.xts` to convert objects of class `timeSeries`, `ts`, `its`, `matrix`, `data.frame`, and `zoo`.

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

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

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

Value

An S3 object of class `xts`.

Author(s)

Jeffrey A. Ryan

See Also

[xts](#)

axTicksByTime

Compute x-Axis Tickmark Locations by Time

Description

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

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

Usage

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

Arguments

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

Details

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

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

Value

A vector of index points to break on, possibly with the index names.

Author(s)

Jeffrey A. Ryan

See Also

[endpoints](#)

Examples

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

CLASS *Extract and Set .CLASS Attribute*

Description

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

Usage

```
CLASS(x)

CLASS(x) <- value
```

Arguments

<code>x</code>	an <code>xts</code> object
<code>value</code>	the new <code>.CLASS</code> value to assign

Details

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

It is best to use the traditional `as` methods.

Value

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

Author(s)

Jeffrey A. Ryan

See Also

[as.xts](#), [reclass](#)

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

Description

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

Usage

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

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

Arguments

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

Details

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

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

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

Value

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

The replacement functions are called for their side-effects.

Author(s)

Jeffrey A. Ryan

See Also

[coredata](#), [xtsAttributes](#)

Examples

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

diff.xts

*Lags and Differences of xts Objects***Description**

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

Usage

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

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

Arguments

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

Details

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

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

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

Value

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

Author(s)

Jeffrey A. Ryan

References

<http://en.wikipedia.org/wiki/Lag>

Examples

```
x <- xts(1:10, Sys.Date()+1:10)
lag(x)
lag(x, k=2)

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

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

dimnames.xts

Dimnames of an xts Object

Description

Get or set dimnames of an `xts` object.

Usage

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

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

Arguments

<code>x</code>	an <code>xts</code> object
<code>value</code>	a list object of length two. See Details.

Details

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

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

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

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

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

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

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

Value

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

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

Note

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

Author(s)

Jeffrey A. Ryan

See Also

[xts](#)

Examples

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

`endpoints`*Locate Endpoints by Time*

Description

Extract index values of a given `xts` object corresponding to the *last* observations given a period specified by `on`

Usage

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

Arguments

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

Details

`endpoints` returns a numeric vector corresponding to the *last* observation in each period specified by `on`, with a zero added to the beginning of the vector, and the index of the last observation in `x` at the end.

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

Value

A numeric vector of endpoints beginning with 0 and ending with the a value equal to the length of the `x` argument.

Author(s)

Jeffrey A. Ryan

Examples

```
data(sample_matrix)

endpoints(sample_matrix)
endpoints(sample_matrix, 'weeks')
```

 first

 Return First or Last n Elements of A Data Object

Description

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

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

Usage

```

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

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

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

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

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

```

Arguments

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

Details

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

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

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

`n` may be positive or negative, in either numeric or character contexts. When positive it will return the result expected - e.g. `last(X, '1 month')` will return the last month's data. If negative, all

data will be returned *except* for the last month. It is important to note that this is not the same as calling `first(X, '1 month')` or `first(X, '-1 month')`. All 4 variations return different subsets of data and have distinct purposes.

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

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

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

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

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

Value

A subset of elements/rows of the original data.

Author(s)

Jeffrey A. Ryan

Examples

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

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

x <- xts(1:100, Sys.Date()+1:100)
first(x, 10)
first(x, '1 day')
first(x, '4 days')
first(x, 'month')
last(x, '2 months')
last(x, '6 weeks')
```

`firstof`*Create a POSIXct Object*

Description

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

Usage

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

Arguments

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

Details

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

Value

An object of class `POSIXct`.

Author(s)

Jeffrey A. Ryan

See Also

[ISOdatetime](#)

Examples

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

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

indexClass

Extracting/Replacing the Class of an xts Index

Description

Generic functions to extract, replace, and format the class of the index of an xts object.

Usage

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

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

indexClass(x)
indexClass(x) <- value

indexFormat(x)
indexFormat(x) <- value

convertIndex(x, value)
```

Arguments

x	xts object
value	desired new class or format. See details
...	additional arguments (unused)

Details

The main accessor methods to an xts object's index is via the `index` and `index<-` replacement method. The structure of the index internally is now a numeric value corresponding to seconds since the epoch (POSIXct converted to numeric). This change allows for near native-speed matrix subsetting, as well as nearly instantaneous speed subsets by time.

A call to `index` translates to the desired class on-the-fly. The desired index class is stored as an attribute within the xts object. Upon a standard `index` call, this is used to convert the numeric value to the desired class.

It is possible to view and set the class of the time-index of a given xts object via the `indexClass` function.

To retrieve the raw numeric data a new accessor function (and replacement) has been added `.index`. This is primarily for internal use, but may be useful for end-users.

The specified format must be a character string containing one of the following: `Date`, `POSIXct`, `chron`, `yearmon`, `yearqtr` or `timeDate`.

`indexFormat` only manages the manner in which the object is displayed via `print` (also called automatically when the object is returned) and in conversion to other classes such as `matrix`. The valid values for `indexFormat` are the same for `format`, as this is the function that does the conversion internally.

`convertIndex` returns a modified `xts` object, and does *not* alter the original.

Changing the index type may alter the behavior of **xts** functions expecting a different index, as well as the functionality of additional methods. Use with caution.

Author(s)

Jeffrey A. Ryan

indexTZ

Query the TimeZone of an xts object

Description

Get the `TimeZone` of an `xts` object.

Usage

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

Arguments

<code>x</code>	an <code>xts</code> object
<code>...</code>	unused

Details

As of version 0.6-4 all objects carry the time zone under which they were created in a hidden attribute names `.indexTZ`.

Internally, all time indexing is converted to `POSIXct`, seconds since the epoch as defined by a combination of the underlying OS and the `TZ` variable setting at creation. The current implementation of `xts` manages time zone information as transparently as possible, delegating all management to R, which is in turn managed in most instances by the underlying operating system.

During printing, and subsetting by time strings the internal `POSIX` representation is used to identify in human-friendly terms the time at each position.

This is different than previous versions of **xts**, where the index was stored in its native format (i.e. class).

The ability to create an index using any of the supported `timeBased` classes (`POSIXct`, `Date`, `dates`, `chron`, `timeDate`, `yearmon`, `yearqtr`) is managed at the user-interaction point, and the class is merely stored in another hidden attribute names `.indexCLASS`. This is accessible via the `indexClass` and `indexClass(x) <-` functions.

In most cases, all of this makes the subsetting by time strings possible, and also allows for consistent and fast manipulation of the series internally.

Problems may arise when an object that had been created under one TZ (time zone) are used in a session using another TZ. This isn't usually a issue, but when it is a warning is given upon printing or subsetting.

Value

A named vector of length one, giving the objects TZ at creation.

Note

Timezones are a difficult issue to manage. If intraday granularity is not needed, it is often best to set the system TZ to "GMT" or "UTC".

Author(s)

Jeffrey A. Ryan

See Also

[POSIXt](#)

Examples

```
x <- xts(1:10, Sys.Date()+1:10)
indexTZ(x)
str(x)
x
# now set TZ to something different...
## Not run:
Old.TZ <- Sys.getenv("TZ")
Sys.setenv(TZ="America/Chicago")
x
Sys.setenv(TZ=Old.TZ)
## End(Not run)
```

isOrdered

Check If A Vector Is Ordered

Description

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

Usage

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

Arguments

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

Details

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

Value

Logical

Author(s)

Jeffrey A. Ryan

See Also

[is.unsorted](#)

Examples

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

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

merge.xts

Merge xts Objects

Description

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

Usage

```
merge.xts(...,
           all = TRUE,
           fill = NA,
           suffixes = NULL,
           join = "outer",
           retside = TRUE,
           retclass = "xts")
```

Arguments

<code>...</code>	one or more xts objects, or objects coercible to class xts
<code>all</code>	a logical vector indicating merge type
<code>fill</code>	values to be used for missing elements
<code>suffixes</code>	to be added to merged column names
<code>join</code>	type of database join
<code>retside</code>	which side of the merged object should be returned (2-case only)
<code>retclass</code>	object to return

Details

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

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

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

To do something along the lines of `merge.zoo`’s method of joining based on an `all` argument of the same length of the arguments to join, see the example. A solution to provide this automatically is in development.

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

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

Value

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

Note

This is a highly optimized merge, specifically designed for ordered data.

Author(s)

Jeffrey A. Ryan

References

Merge Join Discussion: <http://blogs.msdn.com/craigfr/archive/2006/08/03/687584.aspx>

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)
```

ndays

Number of Periods in Data

Description

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

Usage

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

Arguments

x A time-based object

Details

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

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

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

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

Value

The number of observations for the period type specified

Author(s)

Jeffrey A. Ryan

See Also

[endpoints](#)

Examples

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

ndays(QQQQ)
nweeks(QQQQ)
## End(Not run)
```

period.apply

Apply Function Over Specified Interval

Description

Apply a specified function to data over a given interval, where the interval is taken to be the data from `INDEX[k]` to `INDEX[k+1]`, for `k=1:(length(INDEX)-1)`.

Usage

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

Arguments

<code>x</code>	data to apply FUN to
<code>INDEX</code>	numeric vector specifying indexing
<code>FUN</code>	an argument of type <code>function</code>
<code>...</code>	additional arguments for FUN

Details

Similar to the rest of the apply family, calculate a specified functions value given a shifting set of data values. The primary difference is that it is that `period.apply` applies a function to non-overlapping intervals along a vector.

Useful for applying arbitrary functions over an entire data object by an arbitrary index, as when INDEX is the result of a call to `breakpoints`.

Value

A vector with length of INDEX minus 1

Author(s)

Jeffrey A. Ryan

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) sd(x))
period.apply(zoo.data, INDEX=ep, FUN=sd) #same

glue <- function(x) { paste(x, collapse='') }
period.apply(letters, c(0, 5, 7, 26), glue)
```

period.max

Calculate Max By Period

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

Author(s)

Jeffrey A. Ryan

See Also

[endpoints](#), [period.sum](#), [period.min](#), [period.prod](#)

Examples

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

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

period.min

Calculate Min By Period

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

Author(s)

Jeffrey A. Ryan

See Also

[endpoints](#), [period.sum](#), [period.max](#), [period.prod](#)

Examples

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

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

period.prod

Calculate Product By Period

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

Author(s)

Jeffrey A. Ryan

See Also

[endpoints](#), [period.sum](#), [period.min](#), [period.max](#)

Examples

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

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

period.sum

Calculate Sum By Period

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

Author(s)

Jeffrey A. Ryan

See Also

[endpoints](#), [period.max](#), [period.min](#), [period.prod](#)

Examples

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

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

periodicity

Approximate Series Periodicity

Description

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

Usage

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

Arguments

<code>x</code>	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 number of days between observations as a `difftime` object, the numerical difference, the units of measurement, and the derived scale of the data as a string.

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

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

Possible `scale` values are:

‘minute’, ‘hourly’, ‘daily’, ‘weekly’, ‘monthly’, ‘quarterly’, and ‘yearly’.

Value

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

Note

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

Author(s)

Jeffrey A. Ryan

See Also

[difftime](#)

Examples

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

plot.xts

Plotting xts Objects

Description

Plotting methods for xts objects.

Usage

```
## S3 method for class 'xts':
plot(x, y = NULL,
      type = "l",
      auto.grid = TRUE,
      major.ticks = "auto",
      minor.ticks = TRUE,
      major.format = TRUE,
      bar.col = "grey",
      candle.col = "white",
      ann = TRUE, axes = TRUE, ...)
```

Arguments

<code>x</code>	an <code>xts</code> object
<code>y</code>	an <code>xts</code> object or <code>NULL</code>
<code>type</code>	type of plot to produce
<code>auto.grid</code>	should grid lines be drawn
<code>major.ticks</code>	should major tickmarks be drawn and labeled
<code>minor.ticks</code>	should minor tickmarks be drawn
<code>major.format</code>	passed along to <code>axTicksByTime</code> . See also
<code>bar.col</code>	the color of the bars when <code>type</code> is 'bars' or 'candles'
<code>candle.col</code>	the color of the candles when <code>type</code> is 'candles'
<code>ann</code>	passed 'par' graphical parameter
<code>axes</code>	passed 'par' graphical parameter
<code>...</code>	additional graphical arguments

Details

Mainly used to draw time-series plots with sensible x-axis labels, it can also plot basic OHLC series using `type='candles'` or `type='bars'`.

Better financial plots can be found in the **quantmod** package, though these are generally incompatible with standard R graphics tools.

Value

Plots an `xts` object to the current device.

Author(s)

Jeffrey A. Ryan

Examples

```
data(sample_matrix)
plot(sample_matrix)
plot(as.xts(sample_matrix))
plot(as.xts(sample_matrix), type='candles')
```

rbind.xts

Concatenate Two or More xts Objects by Row

Description

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

Usage

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

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

Arguments

```
...           objects to bind
deparse.level not implemented
```

Details

Implemented in C, these functions bind `xts` objects by row, resulting in another `xts` object. There may be non-unique index values in either the original series, or the resultant series. Identical indexed series are bound in the order of the arguments passed to `rbind`. See examples. All objects must have the same number of columns, as well as be `xts` objects or coercible to such. `rbind` and `c` are aliases. For traditional merge operations, see `merge.xts` and `cbind.xts`.

Value

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

Note

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

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

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

Author(s)

Jeffrey A. Ryan

See Also[merge.xts](#) [rbind](#)**Examples**

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

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

c(x,x)

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

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

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

Description

Simulated 180 observations on 4 variables.

Usage`data(sample_matrix)`**Format**

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

Examples`data(.sample.matrix)`

`split.xts`*Divide into Groups by Time*

Description

Creates a list of xts objects split along time periods.

Usage

```
split.xts(x, f = "months", drop=FALSE, k = 1, ...)
```

Arguments

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

Details

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

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

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

Value

A list of xts objects.

Note

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

Author(s)

Jeffrey A. Ryan

See Also

[endpoints](#), [split.zoo](#), [aggregate.zoo](#)

Examples

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

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

`timeBased`*Check if Class is Time-Based*

Description

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

Usage

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

Arguments

`x` object to test

Details

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

Value

Logical

Author(s)

Jeffrey A. Ryan

Examples

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

timeBased(200701)
```

timeBasedSeq *Create a Sequence or Range of Times*

Description

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

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

Usage

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

```
timeBasedRange(x, ...)
```

Arguments

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

Details

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

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

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

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

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

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

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

To request a yearly series, it is only necessary to use "1999/2008". Alternately, one could request a monthly series (returned by default as class `yearmon`) with "199901/2008" or "1999-01/2008",

or even "1999/2008-01". As the level of granularity increases, so does the resultant sequence granularity - as does its length.

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

Value

A sequence or range of time-based objects.

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

Author(s)

Jeffrey A. Ryan

References

International Organization for Standardization: ISO 8601 <http://www.iso.org>

See Also

`timeBased`, `xts`

Examples

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

to.period

Convert time series data to an OHLC series

Description

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

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

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

Usage

```

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

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

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

```

Arguments

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

Details

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

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

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

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

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

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

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

Value

An object of the original type, with new periodicity.

Note

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

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

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

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

Author(s)

Jeffrey A. Ryan

Examples

```
data(sample_matrix)

samplexts <- as.xts(sample_matrix)

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

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

 xts

Create Or Test For An xts Time-Series Object

Description

Constructor function for creating an extensible time-series object.

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

Usage

```
xts(x = NULL,
    order.by = index(x),
    frequency = NULL,
    unique = TRUE,
    ...)
```

```
is.xts(x)
```

Arguments

<code>x</code>	an object containing the time series data
<code>order.by</code>	a corresponding vector of unique times/dates - must be of a known time-based class. See details.
<code>frequency</code>	numeric indicating frequency of <code>order.by</code> . See details.
<code>unique</code>	should index be checked for unique time-stamps?
<code>...</code>	additional attributes to be added. See details.

Details

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

The first difference in this extension provides for a requirement that the index values not only be unique and ordered, but also must be of a time-based class. Currently acceptable classes include: 'Date', 'POSIXct', 'timeDate', as well as 'yearmon' and 'yearqtr' where the index values remain unique.

This last uniqueness requirement has been relaxed as of version 0.5-0. By setting `unique=FALSE`, only a check that the index is not decreasing is carried out via the `isOrdered` function.

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

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

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

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

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

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

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

Value

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

Note

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

Author(s)

Jeffrey A. Ryan and Josh M. Ulrich

References

zoo:

See Also

[as.xts](#), [reclass](#), [xtsAttributes](#)

Examples

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

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

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

```

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

```

xtsAPI

xts C API Documentation

Description

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

Last modified: 2008-10-22 by Jeffrey A. Ryan Version: 0.5-0 and above

At present the **xts** API has publically available interfaces to the following:

See xts.h and the code in /src for details.

Internal use functions:

```

int 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_xtsIndexClass(x)
SET_xtsIndexClass(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
xts_IndexClassSymbol

```

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

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

Details

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

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

Value

A named list of user settable attributes.

Author(s)

Jeffrey A. Ryan

See Also

[attributes](#)

Examples

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

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

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

xtsInternals

Internal Documentation

Description

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

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

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

Notable changes in direction include the use of time-based indexing, at first 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 R must be character vectors, and as such are inefficient in both storage and conversion. By eliminating the rownames, and providing a numeric index of R internal type `REAL` or `INTEGER`, it is possible to maintain a connection to standard R date and time classes via the `POSIXct` functions, while at the same time maximizing efficiencies in data handling.

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

There is also new work on providing a C-level API to some of the xts functionality to facilitate external package developers to utilize the fast utility routines such as subsetting and merges, without

having to call only from R. Obviously this places far more burden on the developer to not only understand the internal xts implementation, but also to understand all of what is documented for R-internals (and much that isn't). At present the functions and macros available can be found in the 'xts.h' file in the src directory.

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

Author(s)

Jeffrey A. Ryan

[.xts

Extract Subsets of xts Objects

Description

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

Usage

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

Arguments

x	xts object
i	the rows to extract. Numeric, timeBased or ISO-8601 style (see details)
j	the columns to extract, numeric or by name
drop	should dimension be dropped, if possible
...	additional arguments (unused)

Details

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

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

CCYYMMDD HH:MM:SS.ss+

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

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

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

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

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

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

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

Value

An extraction of the original `xts` object.

Author(s)

Jeffrey A. Ryan

References

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

See Also

`xts`, `.parseISO8601`,

Examples

```
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)
```

Index

- *Topic **chron**
 - align.time, 3
 - diff.xts, 12
- *Topic **datasets**
 - sample_matrix, 34
- *Topic **hplot**
 - plot.xts, 31
- *Topic **manip**
 - align.time, 3
 - diff.xts, 12
 - merge.xts, 22
- *Topic **misc**
 - align.time, 3
 - dimnames.xts, 13
 - indexTZ, 20
 - isOrdered, 21
- *Topic **package**
 - xts-package, 1
- *Topic **ts**
 - align.time, 3
- *Topic **utilities**
 - .parseISO8601, 2
 - [.xts, 46
 - apply.monthly, 5
 - as.xts, 6
 - as.xts.methods, 7
 - axTicksByTime, 9
 - CLASS, 10
 - coredata.xts, 11
 - endpoints, 15
 - first, 16
 - firstof, 18
 - indexClass, 19
 - merge.xts, 22
 - ndays, 24
 - period.apply, 25
 - period.max, 26
 - period.min, 27
 - period.prod, 28
 - period.sum, 29
 - periodicity, 30
 - rbind.xts, 33
 - split.xts, 35
 - timeBased, 36
 - timeBasedSeq, 37
 - to.period, 38
 - xts, 41
 - xtsAPI, 43
 - xtsAttributes, 44
 - xtsInternals, 45
 - .dimnames.xts (*xtsInternals*), 45
 - .index (*indexClass*), 19
 - .index<- (*indexClass*), 19
 - .parseISO8601, 2, 47
 - [.xts, 46
- aggregate.zoo, 35
- align.time, 3
- apply.daily (*apply.monthly*), 5
- apply.monthly, 5, 26
- apply.quarterly (*apply.monthly*), 5
- apply.weekly (*apply.monthly*), 5
- apply.yearly (*apply.monthly*), 5
- as.fts.xts (*as.xts.methods*), 7
- as.xts, 2, 6, 11, 42
- as.xts.data.frame
 - (*as.xts.methods*), 7
- as.xts.fts (*as.xts.methods*), 7
- as.xts.its (*as.xts.methods*), 7
- as.xts.matrix (*as.xts.methods*), 7
- as.xts.methods, 7, 7
- as.xts.timeSeries
 - (*as.xts.methods*), 7
- as.xts.ts (*as.xts.methods*), 7
- as.xts.xts (*as.xts.methods*), 7
- as.xts.zoo (*as.xts.methods*), 7
- attributes, 45
- axTicksByTime, 9

- `c.xts` (`rbind.xts`), 33
- `cbind.xts` (`merge.xts`), 22
- CLASS, 10
- CLASS<- (*CLASS*), 10
- `convertIndex` (*indexClass*), 19
- `coredata`, 12
- `coredata.xts`, 11
- `diff.xts`, 12
- `difftime`, 31
- `dimnames.xts`, 13
- `dimnames.xts`<- (*xtsInternals*), 45
- `dimnames`<-.*xts* (*dimnames.xts*), 13
- `endpoints`, 5, 10, 15, 25–30, 35
- `first`, 16
- `firstof`, 18
- `index.xts` (*indexClass*), 19
- `index`<-.*xts* (*indexClass*), 19
- `indexClass`, 19
- `indexClass`<- (*indexClass*), 19
- `indexFormat` (*indexClass*), 19
- `indexFormat`<- (*indexClass*), 19
- `indexTZ`, 20
- `is.timeBased` (*timeBased*), 36
- `is.unsorted`, 22
- `is.xts` (*xts*), 41
- ISO8601 (`.parseISO8601`), 2
- ISOdatetime, 18
- `isOrdered`, 21
- `lag.xts` (*diff.xts*), 12
- `last` (*first*), 16
- `lastof` (*firstof*), 18
- `merge.xts`, 22, 34
- `ndays`, 24
- `nhours` (*ndays*), 24
- `nminutes` (*ndays*), 24
- `nmonths` (*ndays*), 24
- `nquarters` (*ndays*), 24
- `nseconds` (*ndays*), 24
- `nweeks` (*ndays*), 24
- `nyears` (*ndays*), 24
- OHLC (*to.period*), 38
- `parseISO8601` (`.parseISO8601`), 2
- `period.apply`, 5, 25
- `period.max`, 26, 28–30
- `period.min`, 27, 27, 29, 30
- `period.prod`, 27, 28, 28, 30
- `period.sum`, 27, 28, 29, 29
- `periodicity`, 30
- `plot.xts`, 31
- POSIXt, 21
- `rbind`, 34
- `rbind.xts`, 33
- `Reclass` (*as.xts*), 6
- `reclass`, 2, 11, 42
- `reclass` (*as.xts*), 6
- `sample_matrix`, 34
- `split.xts`, 35
- `split.zoo`, 35
- `subset.xts` (*[.xts]*), 46
- `timeBased`, 36, 38
- `timeBasedRange` (*timeBasedSeq*), 37
- `timeBasedSeq`, 37
- `TimeZone` (*indexTZ*), 20
- `to.daily` (*to.period*), 38
- `to.hourly` (*to.period*), 38
- `to.minutes` (*to.period*), 38
- `to.minutes10` (*to.period*), 38
- `to.minutes15` (*to.period*), 38
- `to.minutes3` (*to.period*), 38
- `to.minutes30` (*to.period*), 38
- `to.minutes5` (*to.period*), 38
- `to.monthly`, 5
- `to.monthly` (*to.period*), 38
- `to.period`, 4, 38
- `to.quarterly` (*to.period*), 38
- `to.weekly` (*to.period*), 38
- `to.yearly` (*to.period*), 38
- `try.xts` (*as.xts*), 6
- `use.reclass` (*as.xts*), 6
- `use.xts` (*as.xts*), 6
- `xcoredata` (*coredata.xts*), 11
- `xcoredata`<- (*coredata.xts*), 11
- xts*, 2, 7, 9, 14, 38, 41, 47
- xts*-package, 1
- `xtsAPI`, 43
- `xtsAttributes`, 12, 42, 44

`xtsAttributes` <- (`xtsAttributes`),
44
`xtsible` (`as.xts`), 6
`xtsInternals`, 45
`zoo`, 2