

# Package ‘tis’

November 19, 2009

**Title** Time Indexes and Time Indexed Series

**Version** 1.8

**Author** Jeff Hallman <jhallman@frb.gov>

**Depends** R (>= 2.3)

**Enhances** zoo, ggplot2

**Description** Functions and S3 classes for time indexes and time indexed series, which are compatible with FAME frequencies.

**Maintainer** Jeff Hallman <jhallman@frb.gov>

**License** Unlimited

**Date** 2009-11-18 07:08:43

**Repository** CRAN

**Date/Publication** 2009-11-19 13:01:06

## R topics documented:

tis-package	3
aggregate.tis	5
as.data.frame.tis	6
as.Date.jul	7
as.list.keepClass	8
as.list.ti	8
as.matrix.tis	9
as.ts.tis	10
assignList	11
basis	12
between	13
blanks	13
cbind.tis	14
columns	15

constantGrowthSeries . . . . .	15
convert . . . . .	16
csv . . . . .	19
cumsum.tis . . . . .	20
currentMonday . . . . .	20
currentPeriod . . . . .	21
dateRange . . . . .	22
dayOfPeriod . . . . .	23
description . . . . .	25
evalOrEcho . . . . .	25
format.ti . . . . .	26
fortify.tis . . . . .	27
growth.rate . . . . .	29
hms . . . . .	30
holidays . . . . .	31
inferTi . . . . .	32
interpNA . . . . .	33
Intraday . . . . .	34
isIntradayTif . . . . .	35
isLeapYear . . . . .	36
jul . . . . .	36
lags . . . . .	39
latestPeriod . . . . .	40
linearSplineIntegration . . . . .	41
lines.tis . . . . .	43
mergeSeries . . . . .	44
naWindow . . . . .	44
nberShade . . . . .	45
nberShade.ggplot . . . . .	47
plotWindow . . . . .	48
POSIXct . . . . .	49
print.tis . . . . .	50
RowMeans . . . . .	51
scatterPlot . . . . .	52
screenPage . . . . .	56
setDefaultFrequencies . . . . .	58
solve.tridiag . . . . .	59
ssDate . . . . .	60
start.tis . . . . .	61
stripBlanks . . . . .	62
stripClass . . . . .	63
t.tis . . . . .	64
ti . . . . .	64
tiDaily . . . . .	67
tif . . . . .	67
tif2freq . . . . .	69
tis . . . . .	69
tisFilter . . . . .	71

tisFromCsv	72
tisLegend	74
tisPlot	75
today	81
updateColumns	82
window.tis	82
ymd	83

<b>Index</b>	<b>86</b>
--------------	-----------

---

tis-package	<i>Time Indices and Time Indexed Series</i>
-------------	---

---

## Description

Functions and S3 classes for time indices and time indexed series, a flexible kind of time series compatible with series and frequencies understood by the FAME DBMS.

## Details

For a complete list of functions provided by this package, use `library(help="tis")`.

The `ti` (Time Index) and `tis` (Time Indexed Series) classes provide date arithmetic facilities and an alternative to the somewhat inflexible `ts` class in the standard R stats package.

Time Indexes (`ti` class)

A time index has two parts: a `tif` (Time Index Frequency) code and a period. `tif` codes all lie in the open interval (1000..5000) and the period is a nonnegative number less than  $1e10$ . The `ti` encodes both, as for any `ti z`

$$\text{unclass}(z) == (\text{tif}(z) * 1e10) + \text{period}(z)$$

Each `tif` has a particular base period (the time period where `period(z) == 0`). For example, the base period for an "anndecember" (annual December) `ti` is the year ending December 31, 1599. Periods before the base period cannot be represented by instances of the `ti` class.

If `x` and `y` are `ti` objects with the same `tif`, then

$$x - y == \text{period}(x) - \text{period}(y)$$

and

$$x + (\text{period}(y) - \text{period}(x)) == y$$

are both TRUE, so you can use `ti`'s for various calendrical calculations.

A `jul` class is also provided for Julian date-time objects. The `jul()` constructor can create a `jul` from an `ssDate` (spreadsheet date), a `POSIXct` or `POSIXlt` object, a `ti` object, a decimal time (like 2007.5), a `yyyymmdd` number, a `Date`, or anything that can be coerced to a `Date` by `as.Date`. The `ymd()` function and its derivatives (`year()`, `month()`, `day()`, etc.) work on anything that `jul()` can handle.

Time Indexed Series (`tis` class)

The `tis` class maps very closely to the FAME (<http://www.sungard.com/Fame/>) database notion of what a time series is. A `tis` (Time Indexed Series) is vector or matrix indexed by a `ti`. If `x` is

a `tis`, then `start(x)` gives the `ti` for the first observation, and `[start(x) + k]` is the `ti` for the  $k$ 'th observation, while `end(x)` gives the `ti` for the last observation.

You can replace, say, the 5'th observation in a `tis x` by

```
x[start(x) + 4] <- 42
```

and of course the `[]` operator also works with a `ti`. So if you want the value of the daily series `x` from July 3, 1998, you can get it with

```
x[ti(19980703, "daily")]
```

provided, of course, that `ymd(start(x)) <= 19980703 <= ymd(end(x))`.

Numerous methods for `tis` objects are provided:

```
> methods(class = "tis")
 [1] aggregate.tis*      as.data.frame.tis* as.matrix.tis*      as.tis.tis*
 [5] as.ts.tis*          cbind.tis*          cummax.tis*          cummin.tis*
 [9] cumprod.tis*        cumsum.tis*          cycle.tis*           deltat.tis*
[13] diff.tis*           edit.tis*            end.tis*             frequency.tis*
[17] lag.tis*            lines.tis*           Ops.tis*             points.tis*
[21] print.tis*          RowMeans.tis*        RowSums.tis*         start.tis*
[25] tifName.tis*        tif.tis*             time.tis*            [<-.tis*
[29] [.tis*             ti.tis*             t.tis                window.tis*
```

as well as `tisMerge` and `tisPlot` functions. The `convert` function creates series of different frequencies from a given series in ways both more powerful and more flexible than `aggregate` can.

Setting Default Frequencies:

Like the FAME DBMS, the `ti` class has seven weekly frequencies for weeks ending on Sunday, Monday, and it has 12 annual frequencies, for years ending on the last day of each of the 12 months. There are also multiple biweekly, bimonthly and semiannual frequencies.

At any time you can use the function `setDefaultFrequencies` to change which actual frequencies the strings "weekly", "biweekly", "bimonthly", "quarterly", "semiannual" and "annual" refer to. Note (as shown by `args(setDefaultFrequencies)`) that if you don't specify some other frequencies to `setDefaultFrequencies`, you'll get default weeks ending on Monday and default biweeks ending on the second Wednesday. I chose these because they are the weeks and biweeks used most often (for money and reserves data) in my section at the Federal Reserve Board. You might want to use something like

```
setHook(packageEvent("tis", "onLoad"), tis::setDefaultFrequencies(yourArgsGoHere))
```

in your `.First()` to automatically set the defaults up the way you want them whenever this package is loaded.

LAGS:

The `stats::lag(x, k)` function says that a series lagged by a positive  $k$  starts earlier. The opposite is true for the `Lag` function in this package, to maintain consistency with the common usage of 'first lag, second lag' and so on in econometrics.

`tisFilter`:

The `stats::filter` function coerces it's argument to a `ts` time series and returns a `ts`. For `tis` and `zoo` series, this is not right. Both I and the author of the `zoo` package have requested that

`stats::filter` be made an S3 generic with the current version renamed as `filter.default`. This would allow **zoo** and **tis** to define `filter.zoo` and `filter.tis` such that `filter(aZooOrTis)` would do the right thing. We are hopeful that this will happen soon. Meanwhile, `tisFilter` should be used to filter a `tis`.

### Author(s)

Jeff Hallman <jhallman@frb.gov>

Maintainer: ditto

---

aggregate.tis

*Compute Summary Statistics of Time Series Subsets*

---

### Description

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

### Usage

```
## S3 method for class 'tis':
aggregate(x, FUN = sum, ...)
## S3 method for class 'ts':
aggregate(x, nfrequency = 1, FUN = sum, ndeltat = 1,
          ts.eps = getOption("ts.eps"), ...)
```

### Arguments

<code>x</code>	a <code>ts</code> or <code>tis</code> time series.
<code>FUN</code>	a scalar function to compute the summary statistics which can be applied to all data subsets.
<code>nfrequency</code>	new number of observations per unit of time; must be a divisor of the frequency of <code>x</code> .
<code>ndeltat</code>	new fraction of the sampling period between successive observations; must be a divisor of the sampling interval of <code>x</code> .
<code>ts.eps</code>	tolerance used to decide if <code>nfrequency</code> is a sub-multiple of the original frequency.
<code>...</code>	further arguments passed to or used by methods.

## Details

These are time series methods for the generic `aggregate` function.

`aggregate.ts` has been reimplemented in `package:tis` to insure that the resulting time series starts on a boundary of the new frequency. Suppose, for example, that `x` is a monthly series starting in February 1990, `nfrequency` is 4, and `FUN` is `mean`. The `package:tis` implementation will return a series whose first observation is the average of the April, May and June observations of the input series, and the first element of its `tsp` will be 1990.25. The quarters end in March, June, September and December. The first two monthly observations (February and March) are ignored because they don't span a quarter.

The `package:stats` implementation would return a quarterly series whose first observation is the average of the February, March and April monthly observations, and its `tsp` will start with 1990.083, which corresponds to quarters ending in January, April, July and October. In our experience at the Fed, this is not the expected behavior.

`aggregate.ts` and `aggregate.tis` operate similarly. If `x` is not a time series, it is coerced to one. Then, the variables in `x` are split into appropriate blocks of length `frequency(x) / nfrequency`, and `FUN` is applied to each such block, with further (named) arguments in `...` passed to it. The result returned is a time series with frequency `nfrequency` holding the aggregated values.

## See Also

[apply](#), [lapply](#), [tapply](#), [aggregate](#), and [convert](#).

## Examples

```
z <- tis(1:24, start = latestJanuary()) ## a monthly series
aggregate(z, nf = 4, FUN = mean)      ## quarterly average
aggregate(z, nf = 1, FUN = function(x) x[length(x)]) ## December is annual level
```

---

as.data.frame.tis *Coerce to a Data Frame*

---

## Description

Coerce a Time Indexed Series to a data frame.

## Usage

```
## S3 method for class 'tis':
as.data.frame(x, ...)
```

## Arguments

```
x          a tis series
...        other args passed on to as.data.frame.matrix or as.data.frame.vector
```

**Details**

The function is very simple: it calls `as.data.frame.matrix` if `x` is a matrix, or `as.data.frame.vector` if it is not.

**Value**

a data frame.

**See Also**

[data.frame](#)

---

as.Date.jul

*Convert ti or jul objects to Dates*

---

**Description**

Methods to convert `ti` and `jul` objects to class "Date" representing calendar dates.

**Usage**

```
## S3 method for class 'ti':  
as.Date(x, ...)  
## S3 method for class 'jul':  
as.Date(x, ...)
```

**Arguments**

<code>x</code>	A <code>ti</code> or <code>jul</code> object to be converted.
<code>...</code>	Ignored.

**Value**

An object of class "Date".

**See Also**

[as.Date](#) for the generic function, [Date](#) for details of the date class.

**Examples**

```
as.Date(today())           ## invokes as.Date.ti  
as.Date(jul(today() - 7)) ## a week ago, uses as.Date.jul
```

---

```
as.list.keepClass
```

*Convert a vector into a list of objects with the same class*

---

### Description

Turns its argument into a list of elements with the same class as the argument.

### Usage

```
as.list.keepClass(x, ...)
```

### Arguments

<code>x</code>	object to be coerced or tested.
<code>...</code>	objects, possibly named.

### Value

A list `L` of one-element vectors with `L[[i]] == x[i]` for `i` in `1:length(x)`

### Note

The implementation of this function is identical to `as.list.factor`. It is used in `as.list.ti` and `as.list.jul`.

### See Also

[as.list](#)

### Examples

```
as.list.keepClass(today() + 1:5)
```

---

```
as.list.ti
```

*Construct a List from a Time Index or Jul Object*

---

### Description

`as.list.ti` creates a list of one-element `ti` objects from the elements of its arguments.  
`as.list.jul` creates a list of one-element `jul` objects from the elements of its arguments.

### Usage

```
## S3 method for class 'ti':
as.list(x, ...)
## S3 method for class 'jul':
as.list(x, ...)
```

**Arguments**

x                    a ti or jul object  
...                   not used

**Details**

These are the ti and jul methods for the generic as.list.

**Value**

a list of one-element ti or jul objects.

**See Also**

[as.list.keepClass](#), [as.list](#)

**Examples**

```
as.list(today() + 1:5)
as.list(jul(today()) + 1:5)
```

---

as.matrix.tis                    *Create a Matrix from a Time Indexed Series*

---

**Description**

The function adds a dim attribute of c(length(x), 1) to its argument unless it already has a dim attribute of length 2.

**Usage**

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

**Arguments**

x                    a tis object  
...                   ignored

**Value**

A tis object with a dim attribute of length 2.

---

`as.ts.tis`*Convert a Time Indexed Series to a Time Series*

---

### Description

Constructs a `ts` object from a `tis` object. The `tis` object's starting year, starting cycle, and frequency, along with the object's data, in a call to the `ts` function.

### Usage

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

### Arguments

<code>x</code>	a <code>tis</code> object to be converted
<code>...</code>	Ignored

### Details

The `tis` class covers more frequencies than the `ts` class does, so the conversion may not be accurate.

### Value

A `ts` object with the same data as `x`, and with starting time and frequency given by:

```
start = c(year(xstart), cycle(xstart))  
frequency = frequency(x)
```

### Note

The `tis` class covers more frequencies than the `ts` class does, so the conversion may not be accurate.

### See Also

[as.ts](#)

---

assignList	<i>Assign Values In a List to Names</i>
------------	---

---

### Description

Assigns the values in a list to variables in an environment. The variable names are taken from the names of the list, so all of the elements of the list must have non-blank names.

### Usage

```
assignList(aList, pos = -1, envir = as.environment(pos), inherits = FALSE)
```

### Arguments

aList	a list of values to be assigned to variables with names given by names(aList).
pos	where to do the assignment. By default, assigns into the current environment.
envir	the <a href="#">environment</a> to use.
inherits	should the enclosing frames of the environment be inspected?

### Details

See [assign](#) for details on how R assignment works. This function simply uses the elements of names(aList) and aList itself to call the `.Internal` function used by [assign](#) once for each element of aList.

### Value

This function is invoked for its side effect, which assigns values to the variables with names given by names(aList).

### See Also

[assign](#)

### Examples

```
myList <- list(a = 1, b = 2, c = 3)
assignList(myList) ## equivalent to a <- 1; b <- 2; c <- 3
```

---

<code>basis</code>	<i>Optional tis attributes</i>
--------------------	--------------------------------

---

**Description**

`tis` series have (sometimes implicit) `basis` and `observed` attributes, used when aggregating or disaggregating to different frequencies.

**Usage**

```
basis(x)
basis(x) <- value
observed(x)
observed(x) <- value
```

**Arguments**

<code>x</code>	a <code>tis</code> series
<code>value</code>	a character string, see the details

**Details**

These (optional) attributes of a `tis` series are used when converting a series from one frequency to another.

A series `basis` is "business" or "daily", indicating whether the data values in a series are associated with a 5-day business week or a 7-day calendar week.

The `observed` attribute of series is one of the following:

<code>annualized</code>	Specifies that each time series value is the annualized sum of observations made throughout the associated time interval.
<code>averaged</code>	Specifies that each time series value is the average of the observations made throughout the associated time interval.
<code>beginning</code>	Specifies that each time series value represents a single observation made at the beginning of the associated time interval.
<code>end</code>	Specifies that each time series value represents a single observation made at the end of the associated time interval.
<code>formula</code>	Specifies that the time series represents a transformation of other series. For time scale conversion and totaling.
<code>high</code>	Specifies that each time series value is the maximum value for the time interval.
<code>low</code>	Specifies that each time series value is the minimum value for the time interval.
<code>summed</code>	Specifies that each time series value is the sum of observations made throughout the associated time interval.

**Value**

`basis` and `observed` return a character string. The assignment forms invisibly return `x`.

**References**

The FAME documentation, available from Sungard.

**See Also**[convert](#)

---

`between`*Check for Inclusion in a Closed Interval*

---

**Description**

Returns a logical vector like `y` showing if each element lies in the closed interval  $[\min(x1, x2), \max(x1, x2)]$ .

**Usage**

```
between(y, x1, x2)
```

**Arguments**

<code>y</code>	a numeric object
<code>x1</code>	a number
<code>x2</code>	a number

**Value**

A logical object like `y`.

**Examples**

```
mat <- matrix(rnorm(16), 4, 4)
mat
between(mat, -2, 1)
```

---

`blanks`*Blanks*

---

**Description**

Takes an integer argument `n` and return a string of `n` blanks

**Usage**

```
blanks(n)
```

**Arguments**

<code>n</code>	an integer
----------------	------------

---

`cbind.tis`*Combine Series Into a Multivariate (Matrix) Time Indexed Series*

---

### Description

This is `cbind` for `tis` objects. It binds several `ts` and `tis` objects together into a single matrix time indexed series.

### Usage

```
## S3 method for class 'tis':  
cbind(..., union = F)
```

### Arguments

<code>...</code>	any number of univariate or multivariate <code>ts</code> or <code>tis</code> objects. All will be converted to <code>tis</code> objects by <code>as.tis</code> , and the result series all must have the same <code>tif</code> (time index frequency).
<code>union</code>	a logical. If <code>union = F</code> , a matrix created by the intersection of the time windows for the arguments will be created. If <code>union = T</code> , the union of the time windows will be used to create the matrix.

### Details

If `union` is `TRUE` and the series in `...` do not all start and end on the same time index, the missing observations are filled with `NA`.

The column names of the returned series are determined as follows:

If an argument was given a name in the call, the corresponding column has that name. If the argument was itself a matrix with column names, those will be used, otherwise the argument's name is expanded with digits denoting its respective columns.

### Value

a multivariate `tis` object.

### Note

Class `"ts"` has its own `cbind` method which knows nothing about `tis` objects. R generic functions like `cbind` dispatch on the class of their first argument, so if you want to combine `tis` and `ts` objects by calling the generic `cbind`, be sure that the first argument is a `tis`, not a `ts`. You can always ensure this is the case by wrapping the first argument in `...` in `as.tis()`.

### See Also

[cbind](#)

---

columns *Rows and Columns of a Matrix*

---

### Description

Create lists from the rows and/or columns of a matrix.

### Usage

```
columns(z)
rows(z)
```

### Arguments

`z` a matrix

### Value

`rows` returns a list of the rows of `z`. If `z` has row names, those will also be the names of the returned list.

`columns` does the same, but for columns. Note that if `z` is some kind of time series, so too will be the elements of the returned list.

---

constantGrowthSeries  
*Constant Growth Series*

---

### Description

Create `ti`s time series that grow at constant rates.

### Usage

```
fanSeries(startValue, start, end, rates)
tunnelSeries(startValue, start, end, rate, spreads)
```

### Arguments

`startValue` starting value for the series at time `start`

`start` a `ti` (Time Index) for the first observation.

`end` a `ti` or something that can be turned into a `ti` giving the time index for the last observation.

`rates` annual growth rate(s) for the series to be created

`rate` annual growth rate for the series to be created

`spreads` vector of 2 numbers giving the percentage values by which the starting values of the 'tunnel' series should be offset from `startValue`

**Value**

`fanSeries` returns a multivariate series that starts on `start` and ends on `end`. There are `length(rates)` columns. Each column begins at `startValue` and grows at the rate given by its corresponding element in `rates`. These are not true growth rates, rather each column has a constant first difference such that over the course of the first year, column `i` will grow `rates[i]` percent. This yields series that plot as straight lines.

`tunnelSeries` first calls `fanSeries` to create a univariate series running from `start` to `end` with a starting value of `startValue` and growing `rate` percent over the first year. It returns a bivariate series with columns that are offset from that series by `spreads[1]` and `spreads[2]` percent of the `startValue`.

**See Also**

[growth.rate](#)

---

convert

*Time scale conversions for time series*

---

**Description**

Convert `tis` series from one frequency to another using a variety of algorithms.

**Usage**

```
convert(x, tif, method = "constant", observed. = observed(x),
        basis. = basis(x), ignore = F)
```

**Arguments**

<code>x</code>	a univariate or multivariate <code>tis</code> series. Missing values (NAs) are ignored.
<code>tif</code>	a number or a string indicating the desired <code>ti</code> frequency of the return series. See <code>help(ti)</code> for details.
<code>method</code>	method by which the conversion is done: one of "discrete", "constant", "linear", or "cubic". Note that this argument is effectively ignored if <code>observed.</code> is "high" or "low", as the "discrete" method is the only one supported for that setting.
<code>observed.</code>	"observed" attribute of the input series: one of "beginning", "end", "high", "low", "summed", "annualized", or "averaged". If this argument is not supplied and <code>observed(x) != NULL</code> it will be used. The output series will also have this "observed" attribute.
<code>basis.</code>	"daily" or "business". If this argument is not supplied and <code>basis(x) != NULL</code> it will be used. The output series will also have this "basis" attribute.

`ignore` governs how missing (partial period) values at the beginning and/or end of the series are handled. For `method == "discrete"` or `"constant"` and `ignore == T`, input values that cover only part the first and/or last output time intervals will still result in output values for those intervals. This can be problematic, especially for `observed == "summed"`, as it can lead to atypical values for the first and/or last periods of the output series.

## Details

This function is a close imitation of the way FAME handles time scale conversions. See the chapter on "Time Scale Conversion" in the Users Guide to Fame if the explanation given here is not detailed enough.

Start with some definitions. Combining values of a higher frequency input series to create a lower frequency output series is known as *aggregation*. Doing the opposite is known as *disaggregation*.

If `observed == "high"` or `"low"`, the `"discrete"` method is always used.

Disaggregation for `"discrete"` series: (i) for `observed == "beginning"` (`"end"`), the first (last) output period that begins (ends) in a particular input period is assigned the value of that input period. All other output periods that begin (end) in that input period are NA. (ii) for `observed == "high"`, `"low"`, `"summed"` or `"averaged"`, all output periods that end in a particular input period are assigned the same value. For `"summed"`, that value is the input period value divided by the number of output periods that end in the input period, while for `"high"`, `"low"` and `"averaged"` series, the output period values are the same as the corresponding input period values.

Aggregation for `"discrete"` series: (i) for `observed == "beginning"` (`"end"`), the output period is assigned the value of the first (last) input period that begins (ends) in the output period. (ii) for `observed == "high"` (`"low"`), the output period is assigned the value of the maximum (minimum) of all the input values for periods that end in the output period. (iii) for `observed == "summed"` (`"averaged"`), the output value is the sum (average) of all the input values for periods that end in the output period.

Methods `"constant"`, `"linear"`, and `"cubic"` all work by constructing a continuous function  $F(t)$  and then reading off the appropriate point-in-time values if `observed == "beginning"` or `"end"`, or by integrating  $F(t)$  over the output intervals when `observed == "summed"`, or by integrating  $F(t)$  over the output intervals and dividing by the lengths of those intervals when `observed == "averaged"`. The unit of time itself is given by the `basis` argument.

The form of  $F(t)$  is determined by the conversion method. For `"constant"` conversions,  $F(t)$  is a step function with jumps at the boundaries of the input periods. If the first and/or last input periods only partly cover an output period,  $F$  is linearly extended to cover the first and last output periods as well. The heights of the steps are set such that  $F(t)$  aggregates over the input periods to the original input series.

For `"linear"` (`"cubic"`) conversions,  $F(t)$  is a linear (cubic) spline. The x-coordinates of the spline knots are the beginnings or ends of the input periods if `observed == "beginning"` or `"end"`, else they are the centers of the input periods. The y-coordinates of the splines are chosen such that aggregating the resulting  $F(t)$  over the input periods yields the original input series.

For `"constant"` conversions, if `ignore == F`, the first (last) output period is the first (last) one for which complete input data is available. For `observed == "beginning"`, for example, this means that data for the first input period that begins in the first output period is available, while for `observed`

== "summed", this means that the first output period is completely contained within the available input periods. If `ignore == T`, data for only a single input period is sufficient to create an output period value. For example, if converting weekly data to monthly data, and the last observation is June 14, the output series will end in June if `ignore == T`, or May if it is `F`.

Unlike the "constant" method, the domain of  $F(t)$  for "linear" and "cubic" conversions is NOT extended beyond the input periods, even if the ignore option is `T`. The first (last) output period is therefore the first (last) one that is completely covered by input periods.

Series with `observed == "annualized"` are handled the same as `observed == "averaged"`.

## Value

`a` is time series covering approximately the same time span as `x`, but with the frequency specified by `tif`.

## BUGS

Method "cubic" is not currently implemented for `observed "summed"`, `"annualized"`, and `"averaged"`.

## References

Users Guide to `fame`

## See Also

[aggregate](#), [tif](#), [ti](#)

## Examples

```
wSeries <- tis(1:105, start = ti(19950107, tif = "wsaturday"))
observed(wSeries) <- "ending" ## end of week values
mDiscrete <- convert(wSeries, "monthly", method = "discrete")
mConstant <- convert(wSeries, "monthly", method = "constant")
mLinear <- convert(wSeries, "monthly", method = "linear")
mCubic <- convert(wSeries, "monthly", method = "cubic")

## linear and cubic are identical because wSeries is a pure linear trend
cbind(mDiscrete, mConstant, mLinear, mCubic)

observed(wSeries) <- "averaged" ## weekly averages
mDiscrete <- convert(wSeries, "monthly", method = "discrete")
mConstant <- convert(wSeries, "monthly", method = "constant")
mLinear <- convert(wSeries, "monthly", method = "linear")

cbind(mDiscrete, mConstant, mLinear)
```

---

`csv`*Writes a CSV (comma separated values) file.*

---

## Description

Write a matrix or Time Indexed Series to a .csv file that can be imported into a spreadsheet.

## Usage

```
csv(z, file = "", noDates = FALSE, row.names = !is.tis(z), ...)
```

## Arguments

<code>z</code>	matrix or <code>tis</code> object
<code>file</code>	either a character string naming a file or a connection. If "", a file name is constructed by deparsing <code>z</code> . The extension ".csv" is appended to the file name if it is not already there.
<code>noDates</code>	logical. If <code>FALSE</code> (the default) and <code>z</code> is a <code>tis</code> object, the first column of the output file will contain spreadsheet dates.
<code>row.names</code>	either a logical value indicating whether the row names of <code>z</code> are to be written along with <code>z</code> , or a character vector of row names to be written. If <code>FALSE</code> (the default) and <code>z</code> is a <code>tis</code> object, the first column of the output file will contain spreadsheet dates.
<code>...</code>	other arguments passed on to <code>write.table</code> .

## Details

`csv` is essentially a convenient way to call `write.table`. If `file` is not a connection, a file name with the ".csv" extension is constructed. Next, a column of spreadsheet dates is prepended to `z` if necessary, and then `csv` calls

```
write.table(z, file = filename, sep = ",", row.names = !is.tis(z), ...)
```

## Value

`csv` returns whatever the call to `write.table` returned.

## See Also

[write.table](#)

---

`cumsum.tis`*Cumulative Sums, Products, and Extremes*

---

**Description**

Return a `tis` whose elements are the cumulative sums, products, minima or maxima of the elements of the argument.

**Usage**

```
## S3 method for class 'tis':  
cumsum(x)  
## S3 method for class 'tis':  
cumprod(x)  
## S3 method for class 'tis':  
cummax(x)  
## S3 method for class 'tis':  
cummin(x)
```

**Arguments**

`x` a `tis` series.

**Details**

These are `tis` methods for generic functions.

**Value**

A `tis` like `x`. An NA value in `x` causes the corresponding and following elements of the return value to be NA, as does integer overflow in `cumsum` (with a warning).

**See Also**

[cumsum](#), [cumprod](#), [cummin](#), [cummax](#)

---

`currentMonday`*Day of Week Time Indexes*

---

**Description**

Return daily `ti`'s for particular days of the week

**Usage**

```

currentMonday(xTi = today())
currentTuesday(xTi = today())
currentWednesday(xTi = today())
currentThursday(xTi = today())
currentFriday(xTi = today())
currentSaturday(xTi = today())
currentSunday(xTi = today())
latestMonday(xTi = today())
latestTuesday(xTi = today())
latestWednesday(xTi = today())
latestThursday(xTi = today())
latestFriday(xTi = today())
latestSaturday(xTi = today())
latestSunday(xTi = today())

```

**Arguments**

`xTi`                    a `ti` object or something that the `ti()` function can turn into a `ti` object

**Value**

`currentMonday` returns the daily `ti` for the last day of the Monday-ending week that its argument falls into. `currentTuesday` returns the daily `ti` for the last day of the Tuesday-ending week that its argument falls into, and so on for the other weekdays.

`latestMonday` returns the daily `ti` for the last day of the most recent completed Monday-ending week that its argument falls into. Ditto for the other days of the week.

**See Also**

[ti](#)

---

currentPeriod

*Current Period Time Indexes*

---

**Description**

Return a current `ti` of the desired frequency

**Usage**

```

currentWeek(xTi = today())
currentMonth(xTi = today())
currentQuarter(xTi = today())
currentHalf(xTi = today())
currentYear(xTi = today())
currentQ4(xTi = today())

```

```

currentQMonth(xTi = today())
currentJanuary(xTi = today())
currentFebruary(xTi = today())
currentMarch(xTi = today())
currentApril(xTi = today())
currentMay(xTi = today())
currentJune(xTi = today())
currentJuly(xTi = today())
currentAugust(xTi = today())
currentSeptember(xTi = today())
currentOctober(xTi = today())
currentNovember(xTi = today())
currentDecember(xTi = today())

```

### Arguments

`xTi` a `ti` object or something that the `ti()` function can turn into a `ti` object

### Details

`currentWeek` returns the weekly `ti` for the week that its argument falls into. If the argument is itself a `ti`, the returned week contains the last day of the argument's period. The default weekly frequency is "wmonday" (Monday-ending weeks), so `currentWeek` always returns wmonday `ti`'s. This can be changed via the `setDefaultFrequencies` function.

All of the other `current{SomeFreq}` functions work the same way, returning the `ti`'s of `tif` `SomeFreq` that the last day of their arguments period falls into. The `tif`'s for `currentHalf` and `currentQ4` are "semiannual" and "quarterly", respectively. Finally, `currentQMonth` returns the quarter-ending month of the `currentQuarter` of its argument.

`currentJanuary` returns the monthly `ti` for January of the January-ending year that the last day of its argument falls into. `currentFebruary` returns the monthly `ti` for February of the February-ending year that the last day of its argument falls into, and so on.

### Value

All return return `ti` objects as described in the details.

### See Also

[ti](#), [tif](#), [latestWeek](#) [setDefaultFrequencies](#)

---

dateRange

*Start and End Time Indices for a Series*

---

### Description

Returns the starting and ending times of a series in a `ti` object of length 2.

**Usage**

```
dateRange(x)
```

**Arguments**

`x` a `ts` or `tis` time series

**Value**

a `ti` (Time Index) object of length two. The first element is the starting time index, while the second is the ending time index.

**See Also**

[start](#), [end](#), [ti](#), [tis](#)

**Examples**

```
aTs <- ts(1:24, start = c(2001, 1), freq = 12)
aTis <- as.tis(aTs)
dateRange(aTs)
dateRange(aTis)
```

---

dayOfPeriod

*Day positions in Time Index Periods*

---

**Description**

Return position within a `ti` period, or a particular day within the period.

**Usage**

```
dayOfPeriod(xTi = today(), tif = NULL)
dayOfWeek(xTi = today())
dayOfMonth(xTi = today())
dayOfYear(xTi = today())
firstDayOf(xTi)
lastDayOf(xTi)
firstBusinessDayOf(xTi)
lastBusinessDayOf(xTi)
firstBusinessDayOfMonth(xTi)
lastBusinessDayOfMonth(xTi)
currentMonthDay(xTi, daynum)
latestMonthDay(xTi, daynum)
```

**Arguments**

<code>xTi</code>	a <code>ti</code> object or something that the <code>ti()</code> function can turn into a <code>ti</code> object
<code>tif</code>	a time index frequency code or name. See <a href="#">tif</a> .
<code>daynum</code>	day number in month

**Details**

The `dayOfXXXXX` functions all work the same way, returning the day number of the `XXXXX` that `jul(xTi)` falls on. For example, if today is Thursday, January 5, 2006, then `dayOfWeek()`, `dayOfMonth()` and `dayOfYear()` are all 5. All of these are implemented via `dayOfPeriod`, which converts its first argument to a Julian date (via `jul(xTi)`) and finds the `ti` with frequency `tif` that day falls into. It returns the day number of the period represented by that time index that the Julian date falls on.

`firstDayOf` and `lastDayOf` return a daily `ti` for the first or last day of the period represented by `xTi`. `firstBusinessDayOf` and `lastBusinessDayOf` do the same but the returned `ti` has business daily frequency.

`firstBusinessDayOfMonth` returns a business daily `ti` for the first business day of the month of `xTi`. `lastBusinessDayOfMonth` does the same but for the last business day of the month of `xTi`.

`currentMonthDay` returns a daily `ti` for the next upcoming `daynum`'th of the month. `latestMonthDay` does the same for the most recent `daynum`'th of the month.

`currentMonday` returns the daily `ti` for the last day of the Monday-ending week that its argument falls into. The other `current{Weekday}` functions work the same way.

**Value**

All of the functions except the `dayOfXXXXX` return `ti` objects as described in the details section above. The `dayOfXXXXX` functions return numbers.

**Note**

None of these business-day functions take account of holidays, so `firstBusinessDayOfMonth(20010101)`, for example, returns January 1, 2001 which was actually a holiday. To see how to handle holidays, look at the [holidays](#) and [nextBusinessDay](#) help pages.

**See Also**

[ti](#), [tif](#), [jul](#), [holidays](#), [nextBusinessDay](#), [previousBusinessDay](#)

---

description	<i>Description and Documentation Attributes</i>
-------------	---

---

**Description**

Get or set the `description` and `documentation` strings for an object.

**Usage**

```
description(x)
description(x) <- value
documentation(x)
documentation(x) <- value
```

**Arguments**

<code>x</code>	object whose <code>description</code> or <code>documentation</code> attribute is to be set or retrieved
<code>value</code>	a string

**Value**

The setters invisibly return `x`, the getters return the desired attribute or `NULL`.

---

evalOrEcho	<i>Evaluate an Argument If Possible</i>
------------	---

---

**Description**

This function parses, evaluates and returns the string given as its first argument. If it can't, the argument itself is returned. Use of `evalOrEcho` to process arguments inside a function can make for more flexible code.

**Usage**

```
evalOrEcho(x, resultMode = NULL, n = 0)
```

**Arguments**

<code>x</code>	a string or other object to attempt to parse and evaluate.
<code>resultMode</code>	a string or <code>NULL</code> . If non- <code>NULL</code> , the evaluation of <code>x</code> is considered to have failed if the resulting object is not of this mode.
<code>n</code>	parent generations to go back. The evaluation is attempted in the environment specified by <code>parent.frame(n)</code> (of the caller).

**Details**

Using this function inside another function to process some of its arguments can be very useful. For example, `seriesPlot` has a number of arguments that specify text labels for headers, subheaders, footnotes, axis labels, and so on. One of those arguments is `sub`, which specifies the subheader. By doing this:

```
sub <- evalOrEcho(sub, resultMode = "character")
```

`seriesPlot` can handle the `sub` argument given in any of these forms:

1. `sub = "This is a simple subtitle".`
2. `sub = c("this is a two", "line subtitle").`
3. `sub = 'c("this is another", "two line subtitle")'.`

**Value**

If `x` is successfully parsed and evaluated, and its mode matches `resultMode` (if supplied), the resulting object is returned. Otherwise, `x` itself is returned.

**See Also**

[try](#)

---

format.ti

*Convert Time Index or Jul to Character*

---

**Description**

`format` formats a jul or time index object for printing. `as.character` for a jul or ti object is essentially just an alias for `format`.

**Usage**

```
## S3 method for class 'ti':
format(x, ..., tz = "")
## S3 method for class 'jul':
format(x, ...)
## S3 method for class 'ti':
as.character(x, ...)
## S3 method for class 'jul':
as.character(x, ...)
```

**Arguments**

<code>x</code>	a jul or ti (time index) object
<code>tz</code>	A timezone specification to be used for the conversion if <code>x</code> has an intraday frequency. System-specific, but "" is the current time zone, and "GMT" is UTC.
<code>...</code>	other args passed on to <code>format.POSIXlt</code> .

**Details**

The `as.character` methods do nothing but call the corresponding `format` methods. `x` is converted to a `POSIXlt` object and then `format.POSIXlt` takes over.

**Value**

a character vector representing `x`

**Note**

`format.POSIXlt` has been modified to understand two additional format symbols in addition to those documented in `link{strftime}`: `"%q"` in the format string is replaced with the quarter number (1 thru 4) and `"%N"` is replaced with the first letter of the month name.

**See Also**

`format.POSIXlt`, `strftime`

**Examples**

```
format(today() + 0:9, "%x")
as.character(jul(today()))
```

---

fortify.tis

*Fortify a tis object*

---

**Description**

A fortify method for `tis` objects

**Usage**

```
## S3 method for class 'tis':
fortify(x, offset = 0.5, dfNames = NULL, timeName = "date")
```

**Arguments**

<code>x</code>	A <code>tis</code> object of time series
<code>offset</code>	A number between 0 and 1 specifying where in the period of time represented by the <code>'ti(x)'</code> the points should eventually be plotted in <code>ggplot2</code> . <code>'offset = 0'</code> gives the beginning of the period and <code>'offset = 1'</code> the end of the period, <code>'offset = 0.5'</code> the middle of the period, and so on. For example if <code>x</code> is a <code>tis</code> object of quarterly time series and <code>offset = 0.5</code> then the resulting plotted points would fall in the middle of each quarter. <code>offset</code> is passed on to <code>POSIXct(ti(x), offset=offset)</code> and used to create the field <code>date</code> in the resulting data frame.

dfNames	A character vector of the names for the <code>tis</code> objects contained in <code>x</code> . Defaults to the name of the <code>tis</code> object in the univariate case and the column names of the <code>tis</code> object in the multivariate case.
timeName	A character vector of length one with the desired name for the column of dates that will be created from the <code>tis</code> object time index. Default name is "date".

## Details

This function turns a `tis` object into a data frame containing the original time series plus a field of dates adjusted by an 'offset', so that the time series can be more easily plotted with `ggplot2`.

## Author(s)

Trevor Davis

## See Also

[fortify](#)

## Examples

```
if(require("ggplot2") & require("reshape")) {
  # Examples of plotting tis series with ggplot2
  require("datasets")

  # univariate example
  num_discoveries <- as.tis(discoveries)
  ggplot(data = fortify(num_discoveries, offset=0)) +
    geom_line(aes(x=date, y=num_discoveries)) +
    scale_x_date(major="10 years")

  # multivariate example using the "melt trick"
  Seatbelts.tis <- as.tis(Seatbelts[, c("drivers", "front", "rear")])
  Seatbelt.names <- c("Driver", "Front Seat Passenger", "Back Seat Passenger")
  Seatbelts.df <- fortify(Seatbelts.tis, dfNames = Seatbelt.names,
                        timeName = "Time")
  Seatbelts.dfm <- melt(Seatbelts.df, id.var = "Time", variable_name="type")
  qplot( Time, value, data = Seatbelts.dfm, geom="line",
        group=type, colour=type, linetype=type ) +
    geom_vline(xintercept=as.numeric(as.Date("1983-01-31")),
              colour="black", linetype="dashed") +
    ylab("Monthly Road Casulties in the UK")
}
```

---

growth.rate                      *Growth Rates of Time Series*

---

### Description

Get or set growth rates of a `tis` time series in annual percent terms.

### Usage

```
growth.rate(x, lag = 1, simple = T)
growth.rate(x, start = end(x) + 1, simple = T) <- value
```

### Arguments

<code>x</code>	a <code>tis</code> time series or something that can be turned into one by <code>as.tis</code>
<code>lag</code>	number of lags to use in calculating the growth rate as outlined in the details below
<code>simple</code>	simple growth rates if <code>TRUE</code> , compound growth rates if <code>FALSE</code>
<code>start</code>	the first <code>ti</code> time index for which values of <code>x</code> should be replaced to make <code>growth.rate(x[start]) == value[1]</code> .
<code>value</code>	desired growth rates

### Details

An example: Suppose `x` is a quarterly series, then if `simple` is `TRUE`,

```
growth.rate(x, lag = 3) == 100 * ((x[t]/x[t-3]) - 1) * (4/3)
```

while if `simple` is `FALSE`

```
growth.rate(x, lag = 3) == 100 * ((x[t]/x[t-3])^(4/3) - 1).
```

### Value

`growth.rate(x)` returns a `tis` series of growth rates in annual percentage terms.

Beginning with the observation indexed by `start`,

```
growth.rate(x) <- value
```

sets the values of `x` such that the growth rates in annual percentage terms will be equal to `value`. `x` is extended if necessary. The modified `x` is invisibly returned.

---

`hms`*Hours, Minutes and Seconds from a Time Index or Jul*

---

### Description

Extract the fractional part of a `ti` (time index) or `jul` (julian date) object as a normalized list of hours, minutes, and seconds.

### Usage

```
hms(x)
```

### Arguments

`x` a `jul` or something numeric that can be converted into a `jul` with a fractional part.

### Details

The fractional part of `x` is multiplied by 86400 (the number of seconds in a day) and rounded to get the number of seconds. This is then divided by 3600 to get the number of hours, and the remainder of that is divided by 60 to get the normalized number of minutes. The remainder from the second division is the normalized number of seconds.

### Value

A list with components:

<code>hours</code>	Normalized number of hours
<code>minutes</code>	Normalized number of minutes
<code>seconds</code>	Normalized number of seconds

See the details.

### Note

Support for fractional days in `ti` and `jul` objects is relatively new and untested. There is probably code lurking about that assumes the numeric parts of `ti` and `jul` objects are integers, or even code that may round them to make sure they are integers. The fractional parts of `ti` and `jul` objects may not survive encounters with such code.

### See Also

[ti](#) and [jul](#). Also see [hourly](#) for information on intraday frequencies

**Examples**

```
hms(today() + 0.5)
hms(today())
hms(today() + 43201/86400)
```

---

holidays	<i>Holidays</i>
----------	-----------------

---

**Description**

Functions that know about Federal and FRB (Federal Reserve Board) holidays.

**Usage**

```
nextBusinessDay(x, holidays = NULL, goodFriday = F, board = F, inaug = board)
previousBusinessDay(x, holidays = NULL, goodFriday = F, board = F, inaug = board)
isHoliday(x, goodFriday = F, board = F, inaug = board, businessOnly = T)
isGoodFriday(x)
isEaster(x)
holidays(years, goodFriday = F, board = F, inaug = board, businessOnly = T)
federalHolidays(years, board = F, businessOnly = T)
goodFriday(years)
easter(years)
inaugurationDay(years)
holidaysBetween(startTi, endTi, goodFriday = F, board = F, inaug = board, businessOnly = T)
```

**Arguments**

<code>x</code>	a <code>ti</code> time index, or something that can be turned into one, such as a <code>yyyymmdd</code> number or a <code>Date</code> object.
<code>holidays</code>	a vector of holidays (in <code>yyyymmdd</code> form) to skip over, or <code>NULL</code> . In the latter case, the <code>holidays</code> function is used to determine days to skip over.
<code>goodFriday</code>	if <code>TRUE</code> , consider Good Friday as a holiday. Default is <code>FALSE</code> because Good Friday is not a federal holiday.
<code>board</code>	if <code>TRUE</code> , the Friday preceding a Saturday New Years, Independence, Veterans or Christmas Day is considered a holiday.
<code>inaug</code>	if <code>TRUE</code> , consider the U.S. Presidential Inauguration Day to be a holiday, but only if it falls on a weekday.
<code>businessOnly</code>	if <code>TRUE</code> (the default), ignore Saturday New Years, Independence, Veterans and Christmas Day holidays. Has no effect if <code>board</code> is <code>TRUE</code> , since that moves Saturday holidays to Friday.
<code>years</code>	numeric vector of 4 digit years
<code>startTi</code>	a daily <code>ti</code> time index, or something that can be turned into one
<code>endTi</code>	a daily <code>ti</code> time index, or something that can be turned into one

**Details**

Federal law defines 10 holidays. Four of them, New Years, Independence, Veterans and Christmas, fall on the same date every year. The other six fall on particular days of the week and months (MLK, Presidents, Memorial, Labor, Columbus, and Thanksgiving).

If one of the four fixed-date holidays falls on a Sunday, the federal holiday is celebrated the next day (Monday). If it falls on a Saturday, the preceding day (Friday) is a holiday for the Federal Reserve Board, but not for the Reserve Banks and the banking system as a whole.

Presidential Inauguration day is a Federal holiday only in the DC area, and then only if it falls on a weekday, so it is not included in the holidays returned by `federalHolidays`, but it can be included in several of the other functions by setting the `inaug` argument to `TRUE`.

**Value**

`nextBusinessDay` and `previousBusinessDay` return "business" frequency `ti` objects.

`isHoliday`, `isGoodFriday` and `isEaster` return Boolean vectors as long as `x`.

`easter` and `goodFriday` return numeric vectors of `yyyymmdd` dates of the appropriate holidays for each year in the `years` argument.

`inaugurationDay` returns a numeric vector of `yyyymmdd` dates of U.S. Presidential Inauguration Days, if any, that fall in the years given in the `years` argument.

`federalHolidays` returns a numeric vector of `yyyymmdd` dates for the federal holidays for each year in `years`. The `names` attribute of the returned vector contains the holiday names.

`holidays` returns a vector like `federalHolidays` does. The only difference between the two functions is that `holidays` has the option of including Good Fridays.

`holidaysBetween` returns a vector of `yyyymmdd` dates for holidays that fall within the time spanned by `[startTi, endTi]`.

**Note**

The algorithm for finding Easter dates was found somewhere on the web (I don't remember where) and is unbelievably complex. It would probably be simpler to just celebrate the home opener of the Cleveland Indians instead.

---

inferTi

---

*Create a Time Index to Match a Vector of DateTimes*


---

**Description**

Attempts to find a `tif` (Time Index Frequency) that "fits" the supplied `dateTimes`, and returns a `ti` object that hits the same dates and/or times.

**Usage**

```
inferTi(dateTimes)
```

**Arguments**

`dateTimes` a vector POSIXct object, or something that `as.POSIXct` can coerce into one.

**Value**

a `ti` object as long as the input

**Note**

May fail if there is no `ti` that closely matches the inputs.

**See Also**

[as.ti](#)

**Examples**

```
inferTi(Sys.time() + (1:5)*86400)
```

---

interpNA

*Interpolate missing values in a Time Indexed Series*

---

**Description**

Calls [approxfun](#) or [splinefun](#) to interpolate missing values in a `tis` object.

**Usage**

```
interpNA(x, method = "constant", useTimes = F, offset = 1, rule = 2, f = 0, ...)
```

**Arguments**

<code>x</code>	a <code>tis</code> time series
<code>method</code>	One of <code>c("constant", "linear", "fmm", "natural", "periodic")</code> . Methods "constant" and "linear" call <a href="#">approxfun</a> ; the others call <a href="#">splinefun</a> .
<code>useTimes</code>	if TRUE, use <code>time(x, offset)</code> (the decimal times of <code>x</code> ) as the 'x' part of the (x, y) pairs used for interpolation. If FALSE (the default), use <code>ti(x)</code> (the integer time indices of <code>x</code> ) as the 'x' part of the (x, y) pairs.
<code>offset</code>	if <code>useTimes</code> is TRUE, a number in the range [0,1] telling where in the periods represented by <code>ti(x)</code> to get the points for the 'x' parts of the (x, y) pairs. See the help for <a href="#">jul</a> for a more detailed explanation of this parameter.
<code>rule</code>	For methods "constant" and "linear": an integer describing how interpolation is to take place outside the interval <code>[min(x), max(x)]</code> . If <code>rule</code> is 1 then NAs are returned for such points and if it is 2, the value at the closest data extreme is used.

`f` For `method="constant"` a number between 0 and 1 inclusive, indicating a compromise between left- and right-continuous step functions. If `y0` and `y1` are the values to the left and right of the point then the value is  $y_0 * (1-f) + y_1 * f$  so that `f=0` is right-continuous and `f=1` is left-continuous.

`...` Other arguments passed along to `approxfun` for methods "constant" and "linear".

### Details

Depending on the method specified, a call to either `approxfun` or `splinefun` is constructed with appropriate arguments and executed for each column of `x`. In the call to `approxfun` or `splinefun`, the time indices `ti(x)` (or the decimal times returned by `time(x, offset)`, if `useTimes` is `TRUE`) serve as the 'x' argument and the column values as the 'y' argument.

### Value

A `tis` object like `x` with NA values filled in by interpolated values.

### See Also

[approxfun](#), [splinefun](#), [ti](#)

---

Intraday

*R support for Intraday frequencies*

---

### Description

create `tif` (TimeIndexFrequency) codes for hourly, minutely, and secondly `ti`'s.

### Usage

```
hourly(n = 0)
minutely(n = 0)
secondly(n = 0)
```

### Arguments

`n` number of base periods to skip. That is, `hourly(2)` gives a `tif` code for a series observed every 2nd hour, while both `minutely()` and `minutely(1)` are for a series observed once per minute, `secondly(30)` means every 30 seconds, and so on.

**Details**

The current implementation has `hourly(n) -> 2000 + n`, `minutely(n) -> 3000 + n`, and `secondly(n) -> 4000 + n`. If `n` divides evenly into 3600 for `secondly(n)`, the return code will be the same as `hourly(n/3600)`. For `secondly(n)` and `minutely(n)`, if `n` divides evenly into 60, the return code will be as if `minutely(n/60)` or `hourly(n/60)` had been called, respectively.

For `hourly(n)`, `n` must evenly divide into 24 and be less than 24, i.e., `n` is one of 1, 2, 3, 4, 6, 8, 12. For `minutely(n)`, `n` must be an even divisor of 1440, and less than 720. For `secondly(n)`, `n` must divide evenly into 86400, and be no larger than 960.

**Value**

An integer `tif` code.

**See Also**

[tif](#)

---

`isIntradayTif`      *Check for Intraday Time Index Frequency*

---

**Description**

The intraday frequencies are `hourly(n)`, `minutely(n)` and `secondly(n)`, where `n` is an appropriate integer. Their numeric `tif` codes are between 2000 and 4900, and that is what is actually checked for.

**Usage**

```
isIntradayTif(tif)
```

**Arguments**

`tif`                    a character vector of `tif` names (see [tifName](#)) or a numeric vector of `tif` codes (see [tif](#) to be checked)

**Value**

A logical vector as long as the input indicating which elements are intraday Time Index frequencies.

**Note**

The function does not attempt to verify if the supplied `tif` is actually valid, intraday or not.

**See Also**

[hourly](#), [minutely](#), [secondly](#)

**Examples**

```
isIntradayTif(hourly(6))
isIntradayTif(tif(today()))
isIntradayTif(minutely(30))
```

---

isLeapYear	<i>Check Leap Year</i>
------------	------------------------

---

**Description**

Checks whether or not the elements of its input are leap years.

**Usage**

```
isLeapYear(y)
```

**Arguments**

`y` numeric vector of years

**Details**

`y` is a leap year if it is evenly divisible by 4 *and* either it is not evenly divisible by 100 or it is evenly divisible by 400, i.e.,  $y \% 4 == 0 \ \& \ (y \% 100 != 0 \ | \ y \% 400 == 0)$ .

**Value**

logical vector of same length as `y` indicating whether or not the given years are leap years.

**Examples**

```
isLeapYear(c(1899:2004))
```

---

jul	<i>Julian Date Objects</i>
-----	----------------------------

---

**Description**

The function `jul` is used to create `jul` (julian date) objects, which are useful for date calculations.

`as.jul` and `asJul` coerce an object to class "jul", the difference being that `as.jul` calls the constructor `jul`, while `asJul` simply forces the class of its argument to be "jul" without any checking as to whether or not it makes sense to do so.

`is.jul` tests whether an object inherits from class "jul".

**Usage**

```

jul(x, ...)
## S3 method for class 'Date':
jul(x, ...)
## S3 method for class 'ti':
jul(x, offset = 1, ...)
## S3 method for class 'yearmon':
jul(x, offset = 0, ...)
## S3 method for class 'yearqtr':
jul(x, offset = 0, ...)
## Default S3 method:
jul(x, ...)
as.jul(x, ...)
asJul(x)
is.jul(x)

```

**Arguments**

`x` object to be tested (`is.jul`) or converted into a `jul` object. As described in the details below, the constructor function `jul` can deal with several different kinds of `x`.

`...` other args to be passed to the method called by the generic function. `jul.default` may pass these args to `as.Date`.

`offset` For `jul.ti`, a number in the range  $[0,1]$  telling where in the period represented by `x` to find the day. 0 returns the first day of the period, while the default value 1 returns the last day of the period. For example, if `x` has `tif = "wmonday"` so that `x` represents a week ending on Monday, than any `offset` in the range  $[0, 1/7]$  will return the Tuesday of that week, while `offset` in the range  $(1/7, 2/7]$  will return the Wednesday of that week, `offset` in the range  $(6/7, 1]$  will return the Monday that ends the week, and so on.

`jul.yearmon` and `jul.yearqtr` work on `yearmon` and `yearqtr` objects from **zoo**. Note that the default `offset` for these functions is 0, not 1, as that is how the other index-to-date functions in **zoo** work, i.e, if `ym` is a `yearmon` object, then `as.Date(ym)` and `as.jul(ym)` should give the same date.

**Details**

The `jul`'s for any pair of valid dates differ by the number of days between them. R's `Date` class defines a `Date` as a number of days elapsed since January 1, 1970, but `jul` uses the encoding from the *Numerical Recipes* book, which has Jan 1, 1970 = 2440588, and the code for converting between `ymd` and `jul` representations is a straightforward port of the code from that tome.

Adding an integer to, or subtracting an integer from a `jul` results in another `jul`, and one `jul` can be subtracted from another. Two `jul`'s can also be compared with the operators (`==`, `!=`, `<`, `>`, `<=`, `>=`).

The `jul` class implements methods for a number of generic functions, including `"["`, `as.Date`, `as.POSIXct`, `as.POSIXlt`, `c`, `format`, `max`, `min`, `print`, `rep`, `seq`, `ti`, `time`, `ymd`.

`jul` is a generic function with specialized methods to handle `Date` and `ti` objects.

The default method (`jul.default`) deals with character `x` by calling `as.Date` on it. Otherwise, it proceeds as follows:

If `x` is numeric, `isYmd` is used to see if it could be `yyyymmdd` date, then `isTime` is called to see if `x` could be a decimal time (a number between 1799 and 2200). If all else fails, `as.Date(x)` is called to attempt to create a `Date` object that can then be used to construct a `jul`.

### Value

`is.jul` returns `TRUE` or `FALSE`.

`as.jul` and `asJul` return objects with class `"jul"`.

`jul` constructs a `jul` object like `x`.

`jul` with no arguments returns the `jul` for the current day.

### Note

The return value from `asJul` is not guaranteed to be a valid `jul` object. For example, `asJul("a")` will not throw an error, and it will return the string `"a"` with a class attribute `"jul"`, but that's not a valid julian date.

The Julian calendar adopted by the Roman Republic was not accurate with respect to the rotational position of the Earth around the sun. By 1582 it had drifted ten days off. To fix this, Pope Gregory XIII decreed that the day after October 4, 1582 would be October 15, and that thereafter, leap years would be omitted in years divisible by 100 but not divisible by 400. This modification became known as the Gregorian calendar. England and the colonies did not switch over until 1752, by which time the drift had worsened by another day, so that England had to skip over 11 days, rather than 10.

The algorithms used in `jul2ymd` and `ymd2jul` cut over at the end of October 1582.

### References

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes: The Art of Scientific Computing (Second Edition)*. Cambridge University Press.

### See Also

[jul](#), [ymd](#), [ti](#), [as.Date](#)

### Examples

```
dec31 <- jul(20041231)
jan30 <- jul("2005-1-30")
jan30 - dec31          ## 30
feb28 <- jan30 + 29
jul()                  ## current date
```

lags

*Lag a Time Series***Description**

`lag` creates a lagged version of a time series, shifting the time base forward by a given number of observations. `Lag` does exactly the opposite, shifting the time base backwards by the given number of observations. `lag` and `Lag` create a single lagged series, while `lags` and `Lags` can create a multivariate series with several lags at once.

**Usage**

```
## S3 method for class 'tis':
lag(x, k = 1, ...)

Lag(x, k = 1, ...)
lags(x, lags, name = "")
Lags(x, lags, name = "")
```

**Arguments**

<code>x</code>	A vector or matrix or univariate or multivariate time series (including <code>tis</code> series)
<code>k</code>	The number of lags. For <code>lag</code> , this is the number of time periods that the series is shifted <i>forward</i> , while for <code>Lag</code> it is the number of periods that the series is shifted <i>backwards</i> .
<code>...</code>	further arguments to be passed to or from methods
<code>lags</code>	vector of lag numbers. For code <code>lags</code> , each element gives a number of periods by which <code>x</code> is to be shifted <i>forward</i> , while for <code>Lags</code> , each element gives a number of periods by which <code>x</code> is to be shifted <i>backwards</i> .
<code>name</code>	string or a character vector of names to be used in constructing column names for the returned series

**Details**

Vector or matrix arguments 'x' are coerced to time series.

For `lags`, column names are constructed as follows: If `name` is supplied and has as many elements as `x` has columns, those names are used as the base column names. Otherwise the column names of `x` comprise the base column names, or if those don't exist, the first `ncols(x)` letters of the alphabet are used as base names. Each column of the returned series has a name consisting of the basename plus a suffix indicating the lag number for that column.

**Value**

Both functions return a time series (`ts` or `tis`) object. If the `lags` argument to the `lags` function argument has more than one element, the returned object will have a column for each lag, with NA's filling in where appropriate.

---

latestPeriod	<i>Most Recent Period Time Indexes</i>
--------------	--

---

## Description

Return a `ti` for the most recent period of the desired frequency.

## Usage

```
latestWeek(xTi = today())
latestMonth(xTi = today())
latestQuarter(xTi = today())
latestHalf(xTi = today())
latestYear(xTi = today())
latestQ4(xTi = today())
latestJanuary(xTi = today())
latestFebruary(xTi = today())
latestMarch(xTi = today())
latestApril(xTi = today())
latestMay(xTi = today())
latestJune(xTi = today())
latestJuly(xTi = today())
latestAugust(xTi = today())
latestSeptember(xTi = today())
latestOctober(xTi = today())
latestNovember(xTi = today())
latestDecember(xTi = today())
```

## Arguments

`xTi`                    a `ti` object or something that the `ti()` function can turn into a `ti` object

## Details

The `latest{whatever}` functions are the same as the corresponding `current{whatever}` functions, except that they return the most recent completed `ti` of the desired frequency. A period is considered to be completed on its last day. For example, if today is Thursday, then `latestWedweek()` returns the week that ended yesterday. Yesterday it would have returned the same week, but the day before that (Tuesday) it would have returned the "wwednesday" `ti` for the week that had ended six days before.

`latestWeek` returns the weekly `ti` for the most recently completed week as of `xTi`. If the `xTi` is itself a `ti`, the returned week is the most recently completed week as of the last day of `xTi`. (Note that the default weekly frequency is "wmonday" (Monday-ending weeks), so `latestWeek` always returns "wmonday" `ti`'s.) See [setDefaultFrequencies](#) to change this.

All of the other `latest{SomeFreq}` functions work the same way, returning the `ti`'s for the most recently completed `SomeFreq` as of the last day of `xTi`. The `ti`'s (frequencies) for `latestHalf` and `latestQ4` are "semiannual" and "quarterly", respectively.

latestJanuary returns the monthly `ti` for January of the most recently completed January-ending year that the last day of its argument falls into. latestFebruary returns the monthly `ti` for February of the most recently completed February-ending year that the last day of its argument falls into, and so on.

### Value

All return return `ti` objects as described in the details.

### See Also

`ti`, `tif`, `currentWeek` `setDefaultFrequencies`

---

linearSplineIntegration

*Linear Spline Integration*

---

### Description

`lintegrate` gives the values resulting from integrating a linear spline, while `ilspline` returns linear splines that integrate to given values.

### Usage

```
lintegrate(x, y, xint, stepfun = F, rule = 0)
ilspline(xint, w)
```

### Arguments

<code>x</code>	x coordinates of the linear spline <code>F</code> defined by $(x, y)$
<code>y</code>	y coordinates of the linear spline <code>F</code> defined by $(x, y)$
<code>xint</code>	x intervals, i.e. $[x[1], x[2]]$ is the first interval, $[x[2], x[3]]$ is the second interval, and so on.
<code>stepfun</code>	if <code>TRUE</code> , <code>F</code> is a left-continuous step function. The default ( <code>FALSE</code> ) says <code>F</code> is continuous.
<code>rule</code>	one of <code>{0, 1, NA}</code> to specify the behavior of <code>F</code> outside the range of <code>x</code> . Use zero if <code>F</code> is zero outside the range of <code>x</code> , <code>NA</code> if <code>F</code> is <code>NA</code> outside the range of <code>x</code> , and one if <code>F</code> is to be linearly extended outside the range of <code>x</code> .
<code>w</code>	values the linear spline must integrate to

**Details**

`lintegrate` integrates the linear spline  $F$  defined by  $(x, y)$  over the `xint` intervals. The value of  $F$  outside the range of  $x$  is specified by the `rule` argument:

```
rule == 0 --> F(z) = 0 for z outside the range of x
rule == NA --> F(z) = NA for z outside the range of x
rule == 1 --> F(z) extended for z outside the range of x
```

If `stepfun` is `TRUE`,  $F(z)$  is assumed to be a left-continuous step function and the last value of  $y$  is never accessed.

$(x[i], y[i])$  pairs with `NA` values in either `x[i]` or `y[i]` `NA` are ignored in constructing  $F$ .

`ilspline` finds linear splines that integrate over the  $N$  intervals specified by the monotonically increasing  $N+1$  vector `xint` to the  $N$  values given in `w`. The function finds  $N$ -vectors  $x$  and  $y$  such that:

- (i)  $x[j] = (xint[j-1] + xint[j])/2$ , i.e., the values of  $x$  are the midpoints of the intervals specified by `xint`, and
- (ii) the linear spline that passes through the  $(x[i], y[i])$  pairs (and is extended to `xint[1]` and `xint[N+1]` by linear extrapolation) integrates over each interval  $[xint[j], xint[j+1]]$  to `w[j]`.

In fact, `w` can actually be an  $M$  by  $N$  matrix, in which case the  $y$  found by the function is also an  $M$  by  $N$  matrix, with each column of  $y$  giving the  $y$  coordinates of a linear spline that integrates to the corresponding column of `w`.

**Value**

`lintegrate` returns a vector of length `length(xint) - 1`.

`ilspline` returns a list with components named `'x'` and `'y'`.

**See Also**

[spline](#), [approx](#)

**Examples**

```
w <- 10 + cumsum(rnorm(10))
blah <- ilspline(1:11, w)
ww <- lintegrate(blah$x, blah$y, 1:11, rule = 1)
w - ww ## should be all zeroes (or very close to zero)
```

lines.tis

*Plotting Time Indexed Series***Description**

Plotting methods for `tis` objects

**Usage**

```
## S3 method for class 'tis':
lines(x, offset = 0.5, dropNA = FALSE, ...)
## S3 method for class 'tis':
points(x, offset = 0.5, dropNA = FALSE, ...)
```

**Arguments**

<code>x</code>	a <code>tis</code> (time indexed series) object
<code>offset</code>	a number in the range $[0,1]$ telling where in each period of <code>x</code> to plot the point. 0 means the first second of each period, 1 the last second of the period, and the default 0.5 plots each point in the middle of the time period in which it falls.
<code>dropNA</code>	if <code>TRUE</code> , observations with NA values are dropped before calling <code>lines.default</code> or <code>points.default</code> . See the details for why you might or might not want to do this. The default is <code>FALSE</code> , to match the behavior of <code>lines.default</code> and <code>points.default</code> .
<code>...</code>	other arguments to be passed on to <code>lines.default</code> or <code>points.default</code> .

**Details**

These are fairly simple wrappers around the `lines.default` and `points.default`. For example, `lines.tis` basically does this:

```
lines.default(x = time(x, offset = offset), y = x, ...)
```

and `points.tis` is similar. If `dropNA` is `TRUE`, the observations in `x` that are NA are dropped from the `x` and `y` vectors sent to the `.default` functions. For `points`, this shouldn't matter, since `points.tis` omits points with NA values from the plot.

For `lines` the `dropNA` parameter does make a difference. The help document for `lines` says:

"The coordinates can contain NA values. If a point contains NA in either its `x` or `y` value, it is omitted from the plot, and lines are not drawn to or from such points. Thus missing values can be used to achieve breaks in lines."

Note that if the `type` is one of `c("p", "b", "o")`, the non-NA points are still drawn, but line segments from those points to adjacent NA points are not drawn. If `dropNA = TRUE`, the NA points are dropped before calling `lines.default`, and all of the remaining points will be connected with line segments (unless suppressed by the `type` argument).

**See Also**

[lines](#), [points](#)

---

<code>mergeSeries</code>	<i>Merge Time Indexed Series</i>
--------------------------	----------------------------------

---

**Description**

Merge two time-indexed series using either the levels or the first differences of the second series where the series overlap.

**Usage**

```
mergeSeries(x, y, differences = F)
```

**Arguments**

<code>x, y</code>	<code>tis</code> objects, or objects that can sensibly be coerced to <code>tis</code> by <code>as.tis</code> .
<code>differences</code>	if <code>T</code> , the first differences of series are merged, and then cumulatively summed. The default is <code>F</code> .

**Details**

`x` and `y` must have the same `tif` (ti frequency), and the same number of column (if they are multivariate).

**Value**

A `tis` object series with start and end dates that span those of `x` and `y`. Where the series overlap, values from `y` are used.

**See Also**

[`cbind.tis`](#)

---

<code>naWindow</code>	<i>Exclude NA Observations</i>
-----------------------	--------------------------------

---

**Description**

Windows a `tis` or `ts` time series to cut off leading and trailing NA observations.

**Usage**

```
naWindow(x, union = F)
```

**Arguments**

<code>x</code>	a <code>tis</code> or <code>ts</code> time series
<code>union</code>	see details below

**Details**

For multivariate (multiple columns) series and `union = TRUE`, a row of `x` is considered to be NA if and only if all entries in that row are NA. If `union = FALSE` (the default), a row is considered to be NA if any of its entries is NA.

**Value**

A copy of `x` with leading and trailing NA observations deleted.

**See Also**

[window](#)

---

nberShade

*Plotting NBER Recessions*

---

**Description**

`nberDates` returns a matrix with two columns of `yyyymmdd` dates giving the Start and End dates of recessions fixed by the NBER.

`nber.xy` returns a list of `x` and `y` coordinates that can be fed to [polygon](#) to draw NBER shadings on the current plot. It executes `get("nberDates", pos = 1)()` to do so. This means that if you have defined a local version of `nberDates`, it will be used rather than the one supplied by the `tis` package. If the last row from `nberDates()` has a Start entry but an "NA" End entry and `openShade` is `FALSE`, the returned list will not have coordinates for the last row, but will instead include a `vLine` element that gives the `x` coordinate of the last Start. If `openShade` is `TRUE` (the default), the list includes `x` and `y` coordinates for the most recent recession, using the second element of the horizontal range determined by the `xrange` parameter as its end time.

`nberShade` is a generic method for shading recession areas on the current plot. The default version calls `nber.xy` to get `x` and `y` coordinates for the areas to be shaded and then passes those coordinates along with its own arguments to [polygon](#) to do the shading. It also draws a vertical line at the appropriate location if the list returned by `nber.xy` has a `vLine` element.

`romerLines` draws vertical lines on the current plot at the "Romer and Romer" dates when monetary policy is said to have become contractionary.

**Usage**

```
## Default S3 method:
nberShade(col = grey(0.8), border = FALSE, xpd = FALSE,
          xrange = NULL, openShade = TRUE, ...)
nberDates()
nber.xy(xrange = NULL, openShade = TRUE)
romerLines()
```

**Arguments**

	All passed along to <code>polygon</code> :
	color to shade recessionary periods
<code>border</code>	the default ( <code>FALSE</code> ) omits borders on the shaded regions. <code>TRUE</code> draws borders in the foreground color. Alternatively, specify a border color.
<code>xpd</code>	should clipping take place?
<code>...</code>	other args passed to <code>polygon</code>
<code>xrange</code>	horizontal range over which recession shading should be drawn. The default value <code>NULL</code> uses the entire range of the plot. Note however the <code>tisPlot</code> uses the range of the data, which will generally differ the plot range unless the <code>tisPlot</code> parameters <code>leftExpandBy</code> and <code>rightExpandBy</code> are zero. You can force <code>tisPlot</code> to use the plot range by setting the parameter <code>nberArgs = list(xrange = NULL)</code> .
<code>openShade</code>	governs how <code>nber.xy</code> and consequently <code>nberShade</code> handle the case where the last row of the matrix returned by <code>nberDates</code> has an NA in the "End" column, indicating that the end date of the most recent recession is not yet known. See the description above for details.

**Value**

As described above, `nber.xy` returns a list. The other functions described do not return anything useful.

**Note**

Recessions are dated by the Business Cycle Dating Committee of the National Bureau of Economic Research.

The Romer dates are October 1947, September 1955, December 1968, April 1974, August 1978, October 1979 and December 1988.

**References**

Christina D. Romer and David H. Romer. 1989. "Does Monetary Policy Matter? A New Test in the Spirit of Friedman and Schwartz." *NBER Macroeconomics Annual* 4: 121-170.

Christina D. Romer and David H. Romer. 1994. "Monetary Policy Matters." *Journal of Monetary Economics* 34 (August): 75-88.

National Bureau of Economic Research. <http://www.nber.org>.

**See Also**

[polygon](#), [nberShade.ggplot](#)

**Examples**

```
require("datasets")
plot(presidents, type='n', ylab="Presidents approval rating")
nberShade()
lines(presidents)
```

---

nberShade.ggplot *Plotting NBER Recessions in the ggplot graphics system*

---

**Description**

nberShade.ggplot is a method for shading recession areas in a plot of class ggplot. It calls nberDates to get x coordinates for the areas to be shaded and then passes those coordinates along with its own arguments to geom\_rect to do the shading. It may also draw a vertical line at the appropriate location if openShade is false and its xrange is of an appropriate size.

**Usage**

```
## S3 method for class 'ggplot':
nberShade(gg = ggplot2::last_plot(), fill = "grey80", color = NA, size = 0.5,
          alpha = 0.5, xrange = NULL, openShade = TRUE, ...)
```

**Arguments**

gg	A ggplot object that contains a mapping of y to its data. The default is to use the last ggplot object that was plotted.
fill	color to shade recessionary periods, passed to geom_rect.
color	The default (NA) omits borders on the shaded regions. Alternatively, specify a border color. Passed to geom_rect
size	A number specifying how thick should the border lines be. Passed to geom_rect
alpha	A number specifying how transparent the shaded regions should be. Passed to geom_rect
xrange	Horizontal range over which recession shading should be drawn. The default value NULL uses the entire range of the recession dates. xlim and scale_x_date are convenient ggplot2 functions to afterwards restrict the plot to a more sensible range.
openShade	governs how nberShade handles the case where the last row of the matrix returned by nberDates has an NA in the "End" column, indicating that the end date of the most recent recession is not yet known. If TRUE and xrange has been specified and extends far enough into the future it will replace the NA with xrange[2]. Else if openShade is TRUE and xrange is NULL then it will replace the NA with Sys.Date(). If openShade is FALSE then nberShade will draw a vertical line at the start of the most recent recession.
...	not used, present only to match generic

**Value**

Returns a `ggplot` object.

**Note**

Recessions are dated by the Business Cycle Dating Committee of the National Bureau of Economic Research.

**References**

National Bureau of Economic Research. <http://www.nber.org>.

**See Also**

[geom\\_rect](#), [nberDates](#), [nberShade](#)

**Examples**

```
## Not run:
require("ggplot2")
ggp <- ggplot(aes(x=date, y=100*unemploy/pop), data=economics)

nberShade( ggp ) + xlim(as.Date("1967-01-01"), Sys.Date()) +
  opts(legend.position="none") + geom_line() + theme_bw()

nberShade( ggp, fill = "yellow", color = "pink",
  xrange = c("1969-01-01", "2008-02-01"), openShade = FALSE) +
  opts(legend.position="none") + geom_line(size=1) + theme_bw()

## End(Not run)
```

---

plotWindow

*Set up Coordinates for Graphics Window*

---

**Description**

This is a shortcut call to `plot.window` that has some of the parameters set to different defaults than the usual values.

**Usage**

```
plotWindow(xlim, ylim, log = "", asp = NA, xaxs = "i", yaxs = "i", ...)
```

**Arguments**

<code>xlim, ylim</code>	numeric of length 2, giving the x and y coordinates ranges.
<code>log</code>	character; indicating which axes should be in log scale.
<code>asp</code>	numeric, giving the <b>aspect</b> ratio y/x.
<code>xaxs</code>	style of axis interval calculation for the x-axis
<code>yaxs</code>	style of axis interval calculation for the y-axis
<code>...</code>	further graphical parameters as in <a href="#">par</a> .

**See Also**

[plot.window](#)

---

 POSIXct

*Date-time Constructor Functions*


---

**Description**

Functions to create objects of classes "POSIXlt" and "POSIXct" representing calendar dates and times.

**Usage**

```

POSIXct(x, ...)
POSIXlt(x, ...)
## S3 method for class 'jul':
POSIXct(x, ...)
## S3 method for class 'numeric':
POSIXct(x, tz = "", origin, ...)
## S3 method for class 'ti':
POSIXct(x, offset = 1, ...)
## Default S3 method:
POSIXct(x, ...)
## S3 method for class 'jul':
POSIXlt(x, ...)
## S3 method for class 'ti':
POSIXlt(x, ...)
## Default S3 method:
POSIXlt(x, ...)

```

**Arguments**

<code>x</code>	An object to be converted.
<code>tz</code>	A timezone specification to be used for the conversion, <i>if one is required</i> . System-specific (see <a href="#">time zones</a> ), but "" is the current timezone, and "GMT" is UTC (Universal Time, Coordinated).

origin	a date-time object, or something which can be coerced by <code>as.POSIXct(tz="GMT")</code> to such an object.
offset	a number between 0 and 1 specifying where in the period represented by the <code>ti</code> object $x$ the desired time falls. <code>offset = 1</code> gives the first second of the period and <code>offset = 0</code> the last second, <code>offset = 0.5</code> the middle second, and so on.
...	other args passed to <code>ISOdatetime(POSIXct.jul and POSIXct.ti)</code> , <code>as.POSIXct</code> or <code>as.POSIXlt</code> as appropriate. May include a <code>tz</code> argument as above.

### Details

The default methods `POSIXct.default` and `POSIXlt.default` do nothing but call `as.POSIXct` and `as.POSIXlt`, respectively. The `POSIXct.ti` method can take an `offset` argument as explained above, and the `POSIXct.jul` method can handle `jul` objects with a fractional part. The `ti` and `jul` methods for `POSIXlt` just call the `POSIXct` constructor and then convert it's value to a `POSIXlt` object.

### Value

`POSIXct` and `POSIXlt` return objects of the appropriate class. If `tz` was specified it will be reflected in the "tzone" attribute of the result.

### See Also

`as.POSIXct` and `link{as.POSIXlt}` for the default conversion functions, and [DateTime-Classes](#) for details of the classes.

---

print.tis

*Printing Time Indexed Series*

---

### Description

Print method for time indexed series.

### Usage

```
## S3 method for class 'tis':
print(x, format = "%Y%m%d", matrix.format = F, ...)
```

### Arguments

<code>x</code>	a time indexed series
<code>format</code>	a character string describing how to format the observation times if either <code>x</code> is printed in matrix form. Format strings are detailed in <code>format.ti</code> .
<code>matrix.format</code>	TRUE or FALSE. See details.
...	additional arguments that may be passed along to <code>print.ts</code> . See details.

**Details**

If `matrix.format` is `F` (the default) and `x` is a univariate monthly, quarterly or annual series, printing is accomplished by `print(as.ts(x), ...)`. Otherwise, `x` is printed as a matrix with rownames created by formatting the time indexes of the observations according to the `format` string.

**See Also**

[format.ti](#), [print.ts](#)

**Examples**

```
print(tis(1:31, start = today() - 30), format = "%b %d, %Y")
```

---

RowMeans

*Form Row Sums and Means*

---

**Description**

Form row sums and means for numeric arrays.

**Usage**

```
RowSums(x, ...)
RowMeans(x, ...)
## Default S3 method:
RowSums(x, ...)
## Default S3 method:
RowMeans(x, ...)
## S3 method for class 'tis':
RowSums(x, ...)
## S3 method for class 'tis':
RowMeans(x, ...)
```

**Arguments**

`x` an array of two or more dimensions, containing numeric, complex, integer or logical values, or a numeric data frame, or a `tis` time indexed series

`...` arguments passed along to `rowSums` or `rowMeans`.

**Value**

The `tis`-specific methods return a `tis`.

For other types of `x`, see [rowMeans](#) or [rowSums](#).

**See Also**

[rowMeans](#), and [rowSums](#)

**Examples**

```
mat <- tis(matrix(1:36, ncol = 3), start = latestJanuary())
cbind(mat, rowSums(mat), rowMeans(mat))
```

---

scatterPlot

*Produce high-quality scatter plots*


---

**Description**

Plotting function with scads of options for creating high quality scatter plots. Can be used with screenPage.

**Usage**

```
scatterPlot(x, y,
            plotType = "p",
            lineType = "solid", lineWidth = 1.5,
            plotChar = "*", dataCex = 1,
            color = "black",
            xAxisMin = NULL, xAxisMax = NULL, xExpandBy = 0.04,
            xTicks = 5, xTickLocations = NULL,
            labelXTicks = TRUE, xTickLabels = NULL,
            xCex = 1, xAxisLabel = NULL, labelXAxis = TRUE, xSpace = 4,
            yAxisMin = NULL, yAxisMax = NULL, yExpandBy = 0.04,
            yTicks = 5, yTickLocations = NULL,
            yTickLabels = NULL, labelLeftTicks = FALSE, labelRightTicks = TRUE,
            yCex = 1, extendTopTick = TRUE,
            leftAxisLabel = NULL, rightAxisLabel = NULL,
            labelLeftAxis = TRUE, labelRightAxis = FALSE,
            cex = 1,
            head = NULL, headAlign = 0.5, headCex = 1.5,
            sub = NULL, subCex = 0.85,
            leftTopLabel = NULL, rightTopLabel = NULL, topLabelAlign = 0,
            labCex = 1, leftInsideLabel = NULL, rightInsideLabel = NULL,
            innerOffset = 0.05, innerCex = 0.8,
            foot = NULL, footAlign = 0, footCex = 0.8, footSpace = -1,
            tck = 0.03, axisWidth = 2, boxType = "u",
            leftMargin = -1, rightMargin = -1,
            topMargin = -1, bottomMargin = -1)
```

**Arguments**

- x** the x coordinates of points in the plot. If this is a string, the function evalOrEcho will attempt to evaluate the string to obtain the x coordinates.
- y** the y coordinates of points in the plot. If this is a string, the function evalOrEcho will attempt to evaluate the string to obtain the y coordinates.

plotType	type of plot desired. Values are "p" for points (the default), "l" for lines, "b" for both points and lines (lines miss the points), and "o" for overlaid points and lines.
lineType	character or numeric vector specifying the line type if plotType calls for lines. Default is "solid". For most devices, type 1 is solid, 2 is dotted, 3 and up are a mix of dots and dashes.
lineWidth	default is 1.5.
plotChar	character (or number for plotting symbols – see the help for <a href="#">points</a> for details) to be used for plotting points. Default is "*".
dataCex	cex times this number gives the character expansion factor for the data points. Default is 1.
color	string or number. Default is 1, the device default foreground color.
xAxisMin	minimum value of the x axis. If non-NULL, this overrides the calculation described in xExpandBy.
xAxisMax	maximum value of the x axis. If non-NULL, this overrides the calculation described in xExpandBy.
xExpandBy	calculate xAxisMin and xAxisMax by multiplicatively extending the data range in both directions by this amount. Default value .04 extends the data range by 4% in each direction.
xTicks	number of ticks to draw on x axis at "pretty" locations. Default is 5. This argument is ignored if xTickLocations is non-NULL.
xTickLocations	if non-NULL, a vector of desired tick locations or a string that evaluates to such a vector. The default value NULL lets the setting for xTicks take effect.
labelXTicks	If TRUE, label ticks on the x axis. Default is FALSE.
xTickLabels	character vector of tick labels or NULL (the default). If NULL and labelXTicks is TRUE, labels are constructed from the tick locations. This argument has no effect if labelXTicks is FALSE.
xCex	cex times this number gives the character expansion factor for the x-axis labels. Default is 1.
xAxisLabel	text to appear centered under the x axis. Default value NULL creates a string by deparsing the xargument. This argument is ignored if labelXAxis is FALSE.
labelXAxis	if TRUE (the default), label the x axis according to xAxisLabel.
xSpace	lines of space to set aside directly beneath the x-axis to hold tick, year and/or axis labels. Default is 4. The space created is xSpace times labCex.
yAxisMin	minimum value of the y axis. If non-NULL, this overrides the calculation described in yExpandBy.
yAxisMax	maximum value of the y axis. If non-NULL, this overrides the calculation described in yExpandBy.
yExpandBy	calculate yAxisMin and yAxisMax by multiplicatively extending the data range in both directions by this amount. Default value .04 extends the data range by 4% in each direction.

<code>yTicks</code>	number of ticks to draw on y axis at "pretty" locations. Default is 5. This argument is ignored if <code>yTickLocations</code> is non-NULL.
<code>yTickLocations</code>	if non-NULL, a vector of desired tick locations or a string that evaluates to such a vector. The default value NULL lets the setting for <code>yTicks</code> take effect.
<code>yTickLabels</code>	character vector of tick labels or NULL (the default). If NULL and <code>labelLeftTicks</code> or <code>labelRightTicks</code> is TRUE, labels are constructed from the tick locations. This argument has no effect if <code>labelLeftTicks</code> and <code>labelRightTicks</code> are both FALSE.
<code>labelLeftTicks</code>	If TRUE, label ticks on the left axis. Default is FALSE.
<code>labelRightTicks</code>	If TRUE, label ticks on the right axis. Default is TRUE.
<code>yCex</code>	<code>cex</code> times this number gives the character expansion factor for the left and right axis labels. Default is 1.
<code>extendTopTick</code>	if TRUE (the default) the top tick of the y axes encloses the panel. <code>leftAxisMax</code> and <code>rightAxisMax</code> are increased as necessary to include the top tick for enclosing the panel.
<code>leftAxisLabel</code>	text to appear centered outside the left axis. Default value NULL creates a string by deparsing the <code>y</code> argument. This argument is ignored if <code>labelLeftAxis</code> is FALSE.
<code>labelLeftAxis</code>	if TRUE (the default), label the left axis according to <code>leftAxisLabel</code> .
<code>rightAxisLabel</code>	text to appear centered outside the right axis. Default value NULL creates a string by deparsing the <code>y</code> argument. This argument is ignored if <code>labelRightAxis</code> is FALSE.
<code>labelRightAxis</code>	if TRUE label the right axis according to <code>rightAxisLabel</code> . Default is FALSE
<code>cex</code>	the base character expansion factor by which all of the <code>***cex</code> parameters are scaled. The default setting is the value of <code>par("cex")</code> .
<code>head</code>	text to appear at the top of the figure region, with alignment determined by <code>headAlign</code> . No default.
<code>headAlign</code>	number indicating justification for the strings in <code>head</code> and <code>sub</code> . 0 means left justify, 1 means right justify, 0.5 (the default) means to center the text. Other numbers are a corresponding distance between the extremes.
<code>headCex</code>	<code>cex</code> times this number gives the character expansion factor for <code>head</code> . Default is 1.5.
<code>sub</code>	text to appear just under <code>head</code> , with alignment determined by <code>headAlign</code> . No default.
<code>subCex</code>	<code>cex</code> times this number gives the character expansion factor for <code>sub</code> . Default is 0.85.
<code>leftTopLabel</code>	text to appear at the top of the left axis, with alignment determined by <code>topLabelAlign</code> . No default.

<code>rightTopLabel</code>	text to appear at the top of the right axis, with alignment determined by <code>topLabelAlign</code> . No default.
<code>topLabelAlign</code>	number indicating alignment for the strings in <code>leftTopLabel</code> . 0 (the default) means left justify, 1 means right justify, 0.5 means to center the text. <code>rightTopLabel</code> , if given, is aligned by $1 - \text{topLabelAlign}$ .
<code>labCex</code>	<code>cex</code> times this number gives the character expansion factor for <code>leftTopLabel</code> , <code>rightTopLabel</code> , and <code>xAxisLabel</code> .
<code>leftInsideLabel</code>	text to appear left justified and just inside the upper left corner of the plot region. No default.
<code>rightInsideLabel</code>	text to appear right justified and just inside the upper right corner of the plot region. No default.
<code>innerOffset</code>	number between 0 and 1, a fractional offset for the inside labels. The left edge of <code>leftInsideLabel</code> is offset by this fraction of the x range from the left edge of the plot, as is the right edge of <code>rightInsideLabel</code> from the right edge of the plot.
<code>innerCex</code>	<code>cex</code> times this number gives the character expansion factor for <code>leftInsideLabel</code> and <code>rightInsideLabel</code> .
<code>foot</code>	text to appear at the bottom of the figure region, with alignment determined by <code>footAlign</code> . No default.
<code>footAlign</code>	number indicating justification for the strings in <code>foot</code> . 0 (the default) means left justify, 1 means right justify, 0.5 means to center the text. Other numbers are a corresponding distance between the extremes.
<code>footCex</code>	<code>cex</code> times this number gives the character expansion factor for <code>foot</code> . Default is 0.8.
<code>footSpace</code>	lines of space to set aside directly beneath the space allocated by <code>xSpace</code> to hold footnotes. The space created is <code>footSpace</code> times <code>footCex</code> . Default is <code>length(foot)</code> ; using a higher value will result in extra space in the bottom figure margin.
<code>tck</code>	length of major tick marks in inches. Minor ticks are $2/3$ as long. Default is 0.03.
<code>axisWidth</code>	line width for the axes and box (if any). Default is 2.
<code>boxType</code>	character representing the type of box. Characters "o", "l" (ell), "7", "c" will produce boxes which resemble the corresponding upper-case letters. The value "n" will suppress boxes. The default is "u".
<code>leftMargin</code>	lines of space for the left margin. Default value (-1) figures this out automatically.
<code>rightMargin</code>	lines of space for the right margin. Default value (-1) figures this out automatically.
<code>topMargin</code>	lines of space for the top margin. Default value (-1) figures this out automatically.
<code>bottomMargin</code>	lines of space for the bottom margin. Default value (-1) figures this out automatically.

**Details**

Each of the text items `head`, `sub`, `leftTopLabel`, `rightTopLabel`, `leftInsideLabel`, `rightInsideLabel`, `foot`, `leftAxisLabel`, `rightAxisLabel` and `xAxisLabel` can be given as a string, a collections of strings, or as a string that gets evaluated to one of the first two. Multiple strings are drawn on successive lines.

**Value**

`scatterPlot` invisibly returns a list of class "scatterPlot" and elements named `xy` (a matrix containing `x` and `y` in two columns), `plotType`, `lineType`, `color`, `plotChar`, `lineWidth`, `x` (x coordinate for legend), `y` (y coordinate for legend), `xRange`, `yRange`, `innerCex` and `par`. This list is useful mostly as an argument to `legend`.

**Note**

`scatterPlot` is a companion to `tisPlot`. Both are designed to be driven from a graphical user interface.

**See Also**

`tisPlot`, `evalOrEcho`

---

screenPage

*Page Setup for Plots*

---

**Description**

Places header and footer text items in outer margin of page and splits the screen appropriately. Can also redraw header and footer.

**Usage**

```
screenPage(head = NULL, sub = NULL, foot = NULL,
           date = FALSE, dateFormat = "%x", time = FALSE,
           topLeft = character(0), topRight = character(0),
           headFont = par("font.main"), subFont = par("font.sub"),
           footFont = par("font"),
           cex = 1.001, headCex = 1.5, subCex = 0.85,
           footCex = 0.75, topLeftCex = 0.85, topRightCex = 0.85,
           footAlign = 0,
           leftMargin = 0, rightMargin = leftMargin,
           topMargin = 0, bottomMargin = topMargin)
```

**Arguments**

head	character string or strings to appear centered in the top outer margin of the page. If <code>length(head) &gt; 1</code> , a multi-line main title results.
sub	character string or strings to appear centered just under head.
foot	character string or strings to appear in the bottom outer margin of the page.
date	logical: if TRUE, put the current date in the upper right corner of the page.
dateFormat	strptime-style format to use if date is TRUE. The default formats dates like 11/30/2006. What is actually being formatted is the value returned by <code>Sys.time()</code> , so you can also use time formats here. For example, setting <code>dateFormat = "%c"</code> will create a string like "Thu 30 Nov 2006 02:49:45 PM EST".
time	logical: if TRUE, put the current time in the upper right corner of the page. If date is also true, the time string will be on the line below the date string.
topLeft	character string or strings to appear at top left corner of the page
topRight	character string or strings to appear at top right corner of the page
headFont	font to use in writing the main title in head. The default uses whatever <code>par("font.main")</code> is set to.
subFont	font to use in writing the sub title in sub. The default uses whatever <code>par("font.sub")</code> is set to.
footFont	font to use in writing the footnotes in foot. The default uses whatever <code>par("font")</code> is set to.
cex	number by which all of the other "cex" arguments are scaled.
headCex	number: Character Expansion Factor (cex) for the string(s) in head. The actual cex used will be <code>cex * headCex</code> .
subCex	number: cex for the string(s) in sub. The actual cex used will be <code>cex * subCex</code> .
footCex	number: cex for the string(s) in foot. The actual cex used will be <code>cex * footCex</code> .
topLeftCex	number: cex for the string(s) appearing in the top left corner of the page. The actual cex used for these strings will be <code>cex * topLeftCex</code> .
topRightCex	number: cex for the string(s) appearing in the top right corner of the page, including the time and date stamps. The actual cex used for these strings will be <code>cex * topRightCex</code> .
footAlign	number: justification for the strings in foot. 0 means left justify, 1 means right justify, 0.5 means to center the text. Other numbers are a corresponding distance between the extremes.
leftMargin	left margin of page in inches.
rightMargin	right margin of page in inches. Default is same as leftMargin
topMargin	top margin of page in inches.
bottomMargin	bottom margin of page in inches. Default is same as topMargin

**Details**

`screenPage` first sets aside space for the margins specified by `topMargin`, `bottomMargin`, `leftMargin` and `rightMargin`. Then it figures out how much additional space is needed for the top and bottom outer margin text elements, places them, and then splits the screen in 3, with screen 3 being the middle part of the page. The user is then free either to further subdivide screen 3 (using `split.screen()`) or to use it as is.

On exit, screen 3 is the active screen.

**Value**

This function returns a list of all of its arguments, including default values for arguments that were not supplied. The return is invisible if a graphics device is active.

**See Also**

`split.screen`

**Examples**

```
screenPage(head = "Chart 1", date = TRUE, foot = rep(" ", 4),
           cex = 0.85, headCex = 1)

## then draw charts, possibly after further subdividing the screen
```

---

```
setDefaultFrequencies
```

*Return Known Time Index Frequencies, Change Default Frequencies*

---

**Description**

A `tif` (Time Index Frequency) can usually be set either by code (a number) or by name. `setDefaultFrequencies` sets particular frequencies for the `tif` names "weekly", "biweekly", "bimonthly" (also "bimonth"), "quarterly" (also "q"), "annual" (also "a"), and "semiannual" (also "sann").

`tifList` returns the map of frequency names to frequency codes.

**Usage**

```
setDefaultFrequencies(weekly      = "wmonday",
                      biweekly    = "bw2wednesday",
                      bimonthly   = "bimonthdecember",
                      quarterly    = "qdecember",
                      annual       = "anndecember",
                      semiannual   = "sanndecember",
                      setup = FALSE)

tifList()
```

**Arguments**

weekly	A string giving the name of the particular frequency that frequency "weekly" will correspond to
biweekly	Ditto for "biweekly"
bimonthly	Ditto for "bimonth" and "bimonthly"
quarterly	Ditto for "q" and "quarterly"
annual	Ditto for "a" and "annual"
semiannual	Ditto for "sann" and "semiannual"
setup	If TRUE, set all of the defaults, otherwise set only the defaults for which arguments were given. The default is FALSE, but see the details

**Details**

The named vector `.tifList` (returned by the function of the same name) stored in the global environment contains the mapping of frequency names to frequency codes. Running this function modifies the `tifList` vector and stores it back in the global environment. It gets run with `setup = TRUE` when the `tis` package is loaded. If you want different defaults, call the function sometime after that.

**Value**

A copy of the `.tifList` vector.

**See Also**

[tifName](#)

---

`solve.tridiag`

*Solve a Tridiagonal System of Equations*

---

**Description**

This function solves the equation  $a \begin{smallmatrix} \%*\% \\ x \end{smallmatrix} = b$  for  $x$ , where  $a$  is tridiagonal and  $b$  can be either a vector or a matrix.

**Usage**

```
## S3 method for class 'tridiag':
solve(a, b, ...)
```

**Arguments**

a	a <code>tridiag</code> object: a square tridiagonal (all zeroes except for the main diagonal and the diagonals immediately above and below it) matrix containing the coefficients of the linear system.
b	a vector or matrix giving the right-hand side(s) of the linear system. If missing, b is taken to be an identity matrix and the function will return the inverse of a.
...	ignored

**Details**

Uses the LINPACK `dgt sv` routine.

**See Also**

[solve](#)

---

ssDate

*ssDate Objects*

---

**Description**

The function `ssDate` is used to create `ssDate` (spreadsheet date) objects, which are useful for reading and writing dates in spreadsheet form, i.e., as the number of days since December 30, 1899.

`as.ssDate` and `is.ssDate` coerce an object to a `ssDate` and test whether an object is a `ssDate`.

**Usage**

```
ssDate(x, ...)
as.ssDate(x)
is.ssDate(x)
```

**Arguments**

x	object to be tested ( <code>is.ssDate</code> ) or converted into a <code>ssDate</code> object.
...	other args to be passed to <code>jul</code> function.

**Details**

an `ssDate` is essentially a rebased Julian date that represents a date as the number of days since December 30, 1899. The constructor function `ssDate` subtracts `jul(18991230)` from `jul(x, ...)` and coerces the result to class `ssDate`. Pretty much all of the stuff you can do with `jul` objects can also be done with `ssDate` objects.

**Value**

`is.ssDate` returns TRUE or FALSE.

`as.ssDate` coerces its argument to have class `ssDate`, without making any attempt to discern whether or not this is a sensible thing to do.

`ssDate` constructs a `ssDate` object like `x`.

`ssDate` with no arguments returns the `ssDate` for the current day.

**See Also**

[jul](#)

**Examples**

```
dec31 <- ssDate(20041231)
jan30 <- ssDate("2005-1-30")
jan30 - dec31          ## 30
feb28 <- jan30 + 29
ssDate()              ## current date
```

---

start.tis

*Starting and ending time indexes*

---

**Description**

Return the start or end time index for a `tis` object.

**Usage**

```
## S3 method for class 'tis':
start(x, ...)
## S3 method for class 'tis':
end(x, ...)
start(x) <- value
```

**Arguments**

<code>x</code>	a <code>tis</code> object
<code>value</code>	desired start attribute
<code>...</code>	ignored

**Value**

`start.tis` returns the start attribute of `x`, while `end.tis` returns `start(x) + nobs(x) - 1`. `start(x) <- value` returns the series `x` shifted such that its starting time is `value`.

**Note**

`start` and `end` are generic functions with default methods that assume `x` has (or can be given) a `tsp` attribute. The default methods return a two vector as `c(year, period)`, while the methods described here return infinitely more useful `ti` objects.

**See Also**

[start,end](#)

**Examples**

```
x <- tis(numeric(8), start = c(2001, 1), freq = 4)
start(x)          ## --> ti object representing 2001Q1
start(as.ts(x))   ## --> c(2001, 1)
```

---

stripBlanks

*Strip Blanks*

---

**Description**

Strips leading and trailing blanks from strings

**Usage**

```
stripBlanks(strings)
```

**Arguments**

`strings`      character vector

**Value**

An object like `strings` with no leading or trailing blanks.

**See Also**

[blanks](#), [gsub](#)

---

stripClass	<i>Remove part of a class attribute</i>
------------	---

---

### Description

An R object may have a class attribute that is a character vector giving the names of classes it inherits from. `stripClass` strips the class `classString` from that character vector. `stripTis(x)` is shorthand for `stripClass(x, "tis")`.

### Usage

```
stripClass(x, classString)
stripTis(x)
```

### Arguments

`x` an object whose class character vector may or may not include `classString`  
`classString` name of class to remove from the inheritance chain

### Value

An object like `x`, but whose class attribute does not include `classString`. If the class attribute less `classString` is empty, `unclass(x)` is returned.

### Note

This function can be useful in functions that return a modified version of one their arguments. For example, the `format.ti` method takes a `ti` (`TimeIndex`) as an argument and returns a character object object 'like' the original argument. The first thing `format.ti(x)` does internally is `z <- stripClass(x, "ti")`. This creates `z` as a copy of `x` but with the difference that `z` no longer inherits from class `ti`. The function then fills in the data elements of `z` with the appropriate strings and returns it. The beauty of this approach is that the returned `z` already has all of the attributes `x` had, except that it no longer inherits from class `ti`. In particular, if `x` was a matrix with `dimnames`, etc., `z` will also have those attributes.

### See Also

[class](#)

---

<code>t.tis</code>	<i>Matrix Transpose</i>
--------------------	-------------------------

---

**Description**

Returns the transpose of `as.matrix(x)`

**Usage**

```
## S3 method for class 'tis':
t(x)
```

**Arguments**

`x` a `tis` object. If `x` is univariate, it will be treated as if it were a single-column matrix, so its transpose will be a single-row matrix.

**Value**

A matrix, see `t`. Note that this is **not** a time series.

**See Also**

`t`, `tis`

**Examples**

```
a <- tis(matrix(1:30, 5,6), start = latestMonth())
a
t(a) ##i.e., a[i, j] == t(a)[j, i] for all i,j, and t(a) is NOT a time series
```

---

<code>ti</code>	<i>Time Index Objects</i>
-----------------	---------------------------

---

**Description**

The function `ti` is used to create time index objects, which are useful for date calculations and as indexes for `tis` (time indexed series).

`as.ti` and `asTi` coerce an object to a time index, the difference being that `as.ti` calls the constructor `ti`, while `asTi` simply forces the class of its argument to be "ti" without any checking as to whether or not it makes sense to do so.

`is.ti` tests whether an object is a time index.

`couldBeTi` tests whether or not `x` is numeric and has all elements within the range expected for a `ti` time index with the given `tif`. If `tif` is `NULL` (the default), the test is whether or not `x` could be a `ti` of *any* frequency. If so, it can be safely coerced to class `ti` by `as.ti`.

**Usage**

```

ti(x, ...)
## S3 method for class 'Date':
ti(x, ...)
## Default S3 method:
ti(x, tif = NULL, freq = NULL, ...)
## S3 method for class 'jul':
ti(x, tif = NULL, freq = NULL, hour = 0, minute = 0, second = 0, ...)
## S3 method for class 'ssDate':
ti(x, ...)
## S3 method for class 'ti':
ti(x, tif = NULL, freq = NULL, ...)
## S3 method for class 'tis':
ti(x, ...)
## S3 method for class 'yearmon':
ti(x, ...)
## S3 method for class 'yearqtr':
ti(x, ...)
as.ti(x, ...)
asTi(x)
is.ti(x)
couldBeTi(x, tif = NULL)

```

**Arguments**

<code>x</code>	object to be tested ( <code>is.ti</code> ) or converted into a <code>ti</code> object. As described in the details below, the constructor function <code>ti</code> can deal with several different kinds of <code>x</code> .
<code>hour</code>	used if and only if <code>tif</code> is an intraday frequency
<code>minute</code>	used if and only if <code>tif</code> is an intraday frequency
<code>second</code>	used if and only if <code>tif</code> is an intraday frequency
<code>...</code>	other args to be passed to the method called by the generic function.
<code>tif</code>	a <code>ti</code> Frequency, given as either a numerical code or a string. <code>tif()</code> with no arguments returns a list of the allowable numerical codes and names. Either <code>tif</code> or <code>freq</code> must be supplied for the variants of <code>ti()</code> .
<code>freq</code>	some <code>tif</code> 's can alternatively be specified by their frequency, such as 1 (annual), 2 (semiannual), 4 (quarterly), 6 (bimonthly), 12 (monthly), 24 (semimonthly), 26 (biweekly), 36 (tenday), 52 (weekly), 262 (business) and 365 (daily). Either <code>tif</code> or <code>freq</code> must be supplied for the variants of <code>ti()</code> .

**Details**

A `ti` has a `tif` (ti Frequency) and a period. The period represents the number of periods elapsed since the base period for that frequency. Adding or subtracting an integer to a `ti` gives another `ti`. Provided their corresponding element have matching `tifs`, the comparison operators `<`, `>`, `<=`, `>=`, `==` all work, and subtracting one `ti` from another gives the number of periods between them. See the examples section below.

The `ti` class implements methods for a number of generic functions, including `"["`, `as.Date`, `as.POSIXct`, `as.POSIXlt`, `c`, `cycle`, `edit`, `format`, `frequency`, `jul`, `max`, `min`, `print`, `rep`, `seq`, `tif`, `tifName`, `time`, `ymd`.

`ti` is a generic function with specialized methods to handle `jul`, `Date`, `ti.tis`, `yearmon` and `yearqtr` objects.

The default method (`ti.default`) deals with character `x` by calling `as.Date` on it. Otherwise, it proceeds as follows:

If `x` is numeric, a check is made to see if `x` could be a `ti` object that has somehow lost its class attribute. Failing that, `isYmd` is used to see if it could be `yyyymmdd` date, then `isTime` is called to see if `x` could be a decimal time (a number between 1799 and 2200). If `x` is of length 2, an attempt to interpret it as a `c(year, period)` pair is made. Finally, if all else fails, `as.Date(x)` is called to attempt to create a `Date` object that can then be used to construct a `ti`.

## Value

`is.ti` and `couldBeTi` return `TRUE` or `FALSE`.

`as.ti` returns a `ti` object.

`asTi` returns its argument with the class attribute set to `"ti"`.

`ti` constructs a `ti` object like `x`, except for two special cases:

1. If `x` is a `tis` series, the return value is a vector time index with elements corresponding to the observation periods of `x`.
2. If `x` is a numeric object of length 2 interpretable as `c(year, period)`, the return value is a single `ti`.

## Note

The `as.Date(x)` call is not wrapped in a try-block, so it may be at the top of the stack when `ti` fails.

The return value from `asTi` is not guaranteed to be a valid `ti` object. For example, `asTi("a")` will not throw an error, and it will return the string `"a"` with a class attribute `"ti"`, but that's not a valid time index.

## See Also

[jul](#), [ymd](#), [tif](#), [tifName](#), [as.Date](#)

## Examples

```
z <- ti(19971231, "monthly") ## monthly ti for Dec 97
is.ti(z)                    ## TRUE
is.ti(unclass(z))          ## FALSE
couldBeTi(unclass(z))      ## TRUE
ymd(z + 4)                  ## 19980430
z - ti(c(1997,6), freq = 12) ## monthly ti for June 1997
ti(z, tif = "wmonday")     ## week ending Monday June 30, 1997
```

---

`tiDaily`*Daily and Business Day Time Indexes*

---

**Description**

Return a daily or business day `ti` corresponding to a specified position within a time index.

**Usage**

```
tiDaily(xTi, offset = 1)
tiBusiness(xTi, offset = 1)
```

**Arguments**

<code>xTi</code>	a <code>ti</code> object or something that the <code>ti()</code> function can turn into a <code>ti</code> object
<code>offset</code>	for <code>ti(xTi)</code> , a number in the range [0,1] telling where in the period represented by <code>x</code> to find the day. 0 means the first day of the period, 1 the last day of the period, and fractional values for in-between day.

**Value**

`tiDaily` converts its first argument to a `jul` using the `offset` provided, and returns a daily `ti` for that day.

`tiBusiness` converts its first argument to a `jul` using the `offset` provided, and returns a "business" `ti` for that day.

**See Also**

[ti](#), [jul](#)

---

`tif`*Time Index Frequencies and Periods*

---

**Description**

Return the `tif` code of an object, the name associated with a `tif` code, the period number of a time index, or the first .

**Usage**

```

tif(x, ...)
## S3 method for class 'ti':
tif(x, ...)
## S3 method for class 'tis':
tif(x, ...)
## S3 method for class 'ts':
tif(x, ...)
## Default S3 method:
tif(x, freq = NULL, ...)
tifName(s)
## Default S3 method:
tifName(s)
## S3 method for class 'ti':
tifName(s)
## S3 method for class 'tis':
tifName(s)
period(z)
basePeriod(x)

```

**Arguments**

<code>x</code>	a <code>ti</code> or <code>tis</code> object, or a string giving a tif name.
<code>freq</code>	numeric. If <code>x</code> is missing, return the <code>tif</code> for this frequency, otherwise ignore.
<code>...</code>	ignored
<code>s</code>	a <code>ti</code> or <code>tis</code> object, or a tif code.
<code>z</code>	a <code>ti</code> object.

**Details**

The `tifList` object associates `tifNames` with `tif` codes. Most functions that call for `tif` argument can take either a `tif` code or a `tif` name.

Both function are generic function with methods for `ti` and `tis` objects, as well as a default method. `tif` also has a method for `ts` objects.

**Value**

`tif` returns the `tif` code for `x`, while `tifName` returns a name for that code. Many of the codes have several names, but only the default one is returned.

`tif` or `tifName` called with no arguments returns a vector of all `tif` codes with names.

`period` returns a vector like `z` giving the number of periods elapsed since the first period defined for its argument's frequency.

`basePeriod` returns the `ti` for the first period defined for `tif(x)`.

**See Also**

[ti](#), [frequency](#)

**Examples**

```
tif()           ## returns a vector of all tif codes
tifName(today()) ## today() returns a ti
period(today())
```

---

tiffreq *Periods Per Year for Time Index Frequencies*

---

**Description**

Returns the frequency of a `ti` object constructed from the current date with the given `tif`.

**Usage**

```
tiffreq(tif)
```

**Arguments**

`tif` a `tifName` or `tif` code

**Value**

a number

**See Also**

[tif](#), [tifName](#), [frequency](#)

**Examples**

```
tiffreq("wmonday")
tiffreq("monthly")
tiffreq(tif(today()))
```

---

tis *Time Indexed Series*

---

**Description**

The function `tis` is used to create time-indexed series objects.

`as.tis` and `is.tis` coerce an object to a time-indexed series and test whether an object is a time-indexed series.

**Usage**

```

tis(data, start = 1, tif = NULL, frequency = NULL, end = NULL)
as.tis(x, ...)
## S3 method for class 'ts':
as.tis(x, ...)
## S3 method for class 'tis':
as.tis(x, ...)
## S3 method for class 'zoo':
as.tis(x, ...)
## Default S3 method:
as.tis(x, ...)
is.tis(x)

```

**Arguments**

<code>data</code>	a numeric vector or matrix of the observed time-series values.
<code>start</code>	the time of the first observation. This can be a <code>ti</code> object, or anything that <code>ti(start, tif = tif, freq = frequency)</code> , can turn into a <code>ti</code> object.
<code>...</code>	other args to be passed to the method called by the generic function. <code>as.tis.default</code> passes <code>x</code> and <code>...</code> to the constructor function <code>tis</code> .
<code>tif</code>	a <code>ti</code> Frequency, given as either a numerical code or a string. <code>tif()</code> with no arguments returns a list of the allowable numerical codes and names.
<code>frequency</code>	As an alternative to supplying a <code>tif</code> , some <code>tifs</code> can alternatively be specified by their frequency, such as 1 (annual), 2 (semiannual), 4 (quarterly), 6 (bimonthly), 12 (monthly), 24 (semimonthly), 26 (biweekly), 36 (tenday), 52 (weekly), 262 (business) and 365 (daily). Many frequencies have multiple <code>tifs</code> associated with them. For example, all of the <code>tifs</code> ( <code>wsunday</code> , <code>wmonday</code> , ..., <code>wsaturday</code> ) have frequency 52. In this case, specifying <code>freq</code> gets you the default weekly <code>tif</code> <code>wmonday</code> .
<code>end</code>	the time of the last observation, specified in the same way as <code>start</code> .
<code>x</code>	object to be tested ( <code>is.tis</code> ) or converted into a <code>tis</code> object. As described in the details below, <code>as.tis</code> can deal with several different kinds of <code>x</code> .

**Details**

The function `tis` is used to create `tis` objects, which are vectors or matrices with class of `"tis"` and a `start` attribute that is a `ti` (time index) object. Time-indexed series are a form of time series that is more flexible than the standard `ts` time series. While observations for a `ts` object are supposed to have been sampled at equispaced points in time, the observation times for a `tis` object are the times given by successive increments of the more flexible time index contained in the series `start` attribute. There is a close correspondence between Fame time series and `tis` objects, in that all of the Fame frequencies have corresponding `tif` codes.

`tis` objects operate much like vanilla R `ts` objects. Most of the methods implemented for `ts` objects have `tis` variants as well. Evaluate `methods(class = "tis")` to see a list of them.

One way or another, `tis` needs to figure out how to create a `start` attribute. If `start` is supplied, the function `ti` is called with it, `tif` and `frequency` as arguments. The same process is repeated for `end` if it was supplied. If only one of `start` and `end` was supplied, the other is inferred from it and the number of observations in data. If both `start` and `end` are supplied, the function `rep` is used to make data the length implied by `end - start + 1`.

`as.tis` is a generic function with specialized methods for other kinds of time series, including zoo series from **zoo**. The fallback default method calls `tis(x, ...)`.

## Value

`tis` and `as.tis` return time-indexed series. `is.tis` returns TRUE or FALSE.

## Note

If the index of a zoo series is a `ti`, the coercion `as.tis.zoo` does is trivial. For other kinds of zoo series, the function `inferTi` tries to figure out a time index that matches the times of the index of the zoo series. This may fail, as there are infinitely more possible kinds of zoo indexes than the finite number of time index frequencies.

## See Also

Compare with `ts`. See `ti` for details on time indexes. `cbind.tis` combines several time indexed series into a multivariate `tis`, while `mergeSeries` merges series, and `convert` and `aggregate` convert series from one frequency to another. `start.tis` and `end.tis` return `ti` objects, while `ti.tis` returns a vector `ti`. There is a print method `print.tis` and several plotting methods, including `lines.tis` and `points.tis`. The `window.tis` method is also sufficiently different from the `ts` one to deserve its own documentation.

## Examples

```
tis(1:48, start = c(2000, 1), freq = 12)
tis(1:48, start = ti(20000101, tif = "monthly")) ## same result
tis(0, start = c(2000,1), end = c(2000,52), tif = "weekly")
```

---

tisFilter

*Linear Filtering on a Time Series*

---

## Description

Applies linear filtering to a univariate `tis` series or to each column separately of a multivariate `tis` series.

## Usage

```
tisFilter(x, ...)
```

**Arguments**

`x` a univariate or multivariate time series.  
`...` arguments passed along to `filter`.

**Value**

A `tis` time indexed series with leading and trailing NA values stripped.

**Note**

If ever the `filter()` function is made generic, as it should be, this function could become the `tis` method for it.

**See Also**

[filter](#)

**Examples**

```
x <- tis(1:100, start = c(2000,1), freq = 12)
tisFilter(x, rep(1, 3))
tisFilter(x, rep(1, 3), sides = 1)
tisFilter(x, rep(1, 3), sides = 1, circular = TRUE)
```

---

tisFromCsv

*Read time series from Comma Separated Values (.csv) file*

---

**Description**

Reads `tis` (Time Indexed Series) from a csv file, returning the series in a list, and optionally storing them in an environment.

**Usage**

```
tisFromCsv(csvFile, dateCol = "date", dateFormat = "%Y%m%d",
           tif = NULL, defaultTif = "business",
           save = F, envir = parent.frame(),
           naNumber = NULL, tolerance = sqrt(.Machine$double.eps), ...)
```

**Arguments**

`csvFile` A file name, connection, or URL acceptable to [read.csv](#). Also see the the rest of this help entry for required attributes of this file.

`dateCol` name of the column holding dates. This column must be present in the file.

`dateFormat` format of the dates in `dateCol`. If the `dateCol` cells contain Excel dates, use `dateFormat == "excel"`. If they are strings, see [strptime](#) for date formats.

<code>tif</code>	time index frequency of the data. If this is <code>NULL</code> (the default), the function tries to infer the frequency from the dates in the <code>ymdCol</code> column.
<code>defaultTif</code>	If the frequency can't be inferred from the dates in the <code>ymdCol</code> column, this <code>tif</code> frequency will be used. This should be a rare occurrence.
<code>save</code>	If true, save the individual series in the environment given by the <code>envir</code> argument. Default is <code>FALSE</code> .
<code>envir</code>	if <code>save == TRUE</code> , the individual series (one per column) are saved in this environment. Default is the frame of the caller.
<code>naNumber</code>	if non- <code>NULL</code> , numbers within <code>tolerance</code> of this number are considered to be NA values. NA strings can be specified by including an <code>na.strings</code> argument as one of the <code>...</code> arguments that are passed along to <code>read.csv</code> .
<code>tolerance</code>	Used to determine whether or not numbers in the file are close enough to <code>naNumber</code> to be regarded as equal to it. The default is about <code>1.48e-08</code> .
<code>...</code>	Additional arguments passed along to the underlying <code>read.csv</code> function.

### Details

**File Requirements:** The csv file must have column names across the top, and everything but the first row should be numeric. There must be as many column names (enclosed in quotes) as there are columns, and the column named by `dateCol` must have dates in the format indicated by `dateFormat`. **The `dateCol` column must be present.**

**Missing (NA) values:** Missing and NA values are the same thing. The underlying `read.csv` has `","` as its default separator and `"NA"` as its default `na.string`, so the rows

```
20051231,,13,,42,NA,
20060131,NA,14,,43,,NA
```

indicate NA values for both the Dec 2005 and Jan 2006 observations of the first, third, fifth and sixth series.

The values in the file are read into a single large `tis` series, with a `tif` (Time Index Frequency) inferred from the first six dates in the `ymd` column. The first date is converted to a `ti` (Time Index) of that frequency and becomes the `start` of the series. Each individual column is then windowed via `naWindow` to strip off leading and trailing NA values, and the resulting series are put into a list with names given by lower-casing the column names from the csv file. If `save` is `TRUE`, the series are also stored in `envir` using those same names.

### Value

A list of `tis` time series, one per column of the csv file. The list is returned invisibly if `save` is `TRUE`.

### See Also

[ti](#), [tis](#), [read.csv](#), [read.table](#)

---

 tisLegend

*Add a legend to a tisPlot or scatterPlot*


---

### Description

The plotting functions `tisPlot` and `scatterPlot` leave an object named `latestPlot` in the frame from which they were called. `tisLegend` uses that object to set legend arguments (which you can override) and sets reasonable defaults for other arguments.

### Usage

```
tisLegend(..., xrel = 0.1, yrel = 0.1, xjust = 0, yjust = 1, boxType = "n", ncol =
```

### Arguments

<code>...</code>	optional arguments to be passed on to <code>legend</code> . These can include <code>x</code> and <code>y</code> arguments to position the legend, or a list with components named <code>x</code> and <code>y</code> , such as the list returned by <code>locator(1)</code> .
<code>xrel, yrel</code>	Optional numbers between 0 and 1 to specify placement relative to the boundaries of the plot.
<code>xjust, yjust, ncol</code>	passed along to <code>legend</code>
<code>boxType</code>	passed along as <code>bty</code> to <code>legend</code>
<code>cex</code>	gets multiplied by the <code>cex</code> from <code>latestPlot</code> and then passed on to <code>legend</code>

### Details

This function is not strictly necessary, in that you could just call `legend` directly. `tisLegend` makes things a bit easier, however, by using the same argument names as `tisPlot` and `scatterPlot` to specify `color`, `lineType`, `plotChar` and `boxType`, rather than the less intuitive `col`, `lty`, `pch` and `bty` names. The `xrel` and `yrel` arguments provide an alternative way to specify legend placement, one that is used by the `ChartMaker` program.

### Value

a list of the arguments that were sent on to `legend`, with class "tisLegend"

### See Also

[legend](#), [tisPlot](#), [scatterPlot](#)

tisPlot

*Plot time indexed series (tis objects)***Description**

tisPlot is a function with dozens of options for creating high quality time series plots. Can be used with screenPage.

**Usage**

```
tisPlot(...,
  leftAxis = TRUE, plotType = "l",
  lineType = "solid", lineWidth = 1.5,
  plotChar = "*", dataCex = 1,
  color = 1, midPoints = TRUE, dropNA = FALSE,
  xAxisMin = NULL, xAxisMax = NULL, xExpandBy = 0.04,
  xTickFreq = "Auto", xTickSkip = 0,
  xUnlabeledTickFreq = "None", xUnlabeledTickSkip = 0,
  xMinorTickFreq = "None", xMinorTickSkip = 0,
  dateFormat = "Auto", xCex = 1,
  midLabels = FALSE, yearLabels = FALSE,
  xAxisLabel = NULL, xSpace = 4, log = FALSE,
  leftAxisMin = NULL, leftAxisMax = NULL, leftExpandBy = 0.04,
  leftTicks = 5, leftTickLocations = NULL,
  labelLeftTicks = FALSE, leftTickLabels = NULL,
  rightAxisMin = NULL, rightAxisMax = NULL, rightExpandBy = 0.04,
  rightTicks = 5, rightTickLocations = NULL,
  labelRightTicks = TRUE, rightTickLabels = NULL,
  yCex = 1, extendTopTick = TRUE,
  cex = 1,
  head = NULL, headAlign = 0.5, headCex = 1.5,
  sub = NULL, subCex = 0.85,
  leftTopLabel = NULL, rightTopLabel = NULL, topLabelAlign = 0,
  labCex = 1,
  leftInsideLabel = NULL, rightInsideLabel = NULL,
  innerLine = 0.5, innerOffset = 0.05, innerCex = 0.8,
  foot = NULL, footColor = "black", footAlign = 0,
  footCex = 0.8, footSpace = -1,
  tck = 0.03,
  axisWidth = 2,
  start = 0, end = 0,
  boxType = "u",
  leftMargin = -1, rightMargin = -1, topMargin = -1, bottomMargin = -1,
  nberShade = FALSE, nberColor = "gray", nberBorder = FALSE, nberArgs = list
```

**Arguments**

...	any number of univariate or multivariate <code>tis</code> series to be plotted. Non- <code>tis</code> arguments will be converted by <code>as.tis</code> .
<code>leftAxis</code>	logical. <code>leftAxis[i] = TRUE</code> means plot the <i>i</i> 'th series against the left axis, otherwise plot it against the right axis.
<code>plotType</code>	type of plot desired. Values are "p" for points, "l" for lines, "b" for both points and lines (lines miss the points), and "o" for overlaid points and lines.
<code>lineType</code>	character or numeric vector specifying the line type for each series. The default is <code>1:nSeries</code> , where <code>nSeries</code> is the number of series being plotted. Normally type 1 is solid, 2 is dotted, 3 and up are a mix of dots and dashes.
<code>lineWidth</code>	numeric vector of line widths for the series. The default value is <code>1.5</code> .
<code>plotChar</code>	vector of characters (or numbers for plotting symbols – see the help for <code>points</code> for details) to be used for plotting points. Default is "*".
<code>dataCex</code>	numeric vector. <code>cex</code> times these numbers give the character expansion factor for the data points. Default is 1.
<code>color</code>	character or numeric vector specifies color for each series. Default is 1, the device default foreground color.
<code>midPoints</code>	logical. <code>midPoints[i] = TRUE</code> aligns the data points of the <i>i</i> 'th series with the middle day of the periods in which they fall, otherwise data points are aligned with the last day of their periods.
<code>dropNA</code>	if TRUE, observations with NA values are dropped before calling <code>lines.default</code> to draw the lines and/or points on the plot. See the details section of the help for <code>lines.tis</code> for why you might or might not want to do this. The default is FALSE.
<code>xAxisMin</code>	minimum value of the x axis. If non-NULL, this overrides the calculation described in <code>xExpandBy</code> .
<code>xAxisMax</code>	maximum value of the x axis. If non-NULL, this overrides the calculation described in <code>xExpandBy</code> .
<code>xExpandBy</code>	calculate <code>xAxisMin</code> and <code>xAxisMax</code> by multiplicatively extending the data range in both directions by this amount. Default value <code>.04</code> extends the data range by 4% in each direction.
<code>xTickFreq</code>	a string like the ones returned by <code>tifName</code> . This argument and <code>xTickSkip</code> jointly specify locations for labeled x axis ticks as follow: (i) find the <code>ti</code> 's of the given frequency that correspond to <code>xAxisMin</code> and <code>xAxisMax</code> , then (ii) including those as endpoints, generate a sequence of every <code>xTickSkip</code> 'th <code>ti</code> between them.  Two special strings can also be given. "none" means no labelled tick marks, while "auto" tries (not always successfully) to come up with reasonable tick locations automatically. "auto" also overrides any <code>xTickSkip</code> setting. The default is "auto".
<code>xTickSkip</code>	a number used with <code>xTickFreq</code> to specify location of labeled tick marks. The default is zero.
<code>xUnlabeledTickFreq</code>	same as <code>xTickFreq</code> , but for unlabeled major tick locations.

xUnlabeledTickSkip	same as xTickSkip, but for unlabeled major tick locations.
xMinorTickFreq	same as xTickFreq, but for minor tick locations.
xMinorTickSkip	same as xTickSkip, but for minor tick locations.
dateFormat	format string for x axis date labels. See <a href="#">strftime</a> for formats. Default value "auto" tries to come up with a reasonable format automatically, for some bounded value of "reasonable".
xCex	cex times this number gives the character expansion factor for the x-axis labels. Default is 1.
midLabels	if TRUE (the default) x axis label are centered between the ticks they label; if FALSE the label alignment depends on the setting of midPoints. If midPoints is TRUE, the labels (which reflect the last day of the period) are centered under the ticks, else they are (almost) right aligned with the ticks. If rotated labels are specified by nonzero xLabelRotationDegrees, tick labels are right aligned with their ticks.
yearLabels	if TRUE place year labels centered under the x ticks they span. The default is FALSE.
xAxisLabel	text to appear centered under the x axis. Must be a single character string, multi-line xAxisLabel is not supported. No default.
xSpace	lines of space to set aside directly beneath the x-axis to hold tick, year and/or axis labels. Default is 4. The space created is xSpace times labCex.
log	if TRUE use log scaling for y axes. Default is FALSE.
leftAxisMin	minimum value of the left axis. If non-NULL, this overrides the calculation described in leftExpandBy.
leftAxisMax	maximum value of the left axis. If non-NULL, this overrides the calculation described in leftExpandBy.
leftExpandBy	calculate leftAxisMin and leftAxisMax by multiplicatively extending the data range of the leftAxis series in both directions by this amount. Default value .04 extends the data range by 4% in each direction.
leftTicks	number of ticks to draw on left axis at "pretty" locations. Default is 5. This argument is ignored if leftTickLocations is non-NULL.
leftTickLocations	if non-NULL, a vector of desired tick locations or a string that evaluates to such a vector. The default value NULL lets the setting for leftTicks take effect.
labelLeftTicks	If TRUE, label ticks on the left axis. Default is FALSE.
leftTickLabels	character vector of tick labels or NULL (the default). If NULL and labelLeftTicks is TRUE, labels are constructed from the tick locations. This argument has no effect if labelLeftTicks is FALSE.
rightAxisMin	minimum value of the right axis. If non-NULL, this overrides the calculation described in rightExpandBy.

<code>rightAxisMax</code>	maximum value of the right axis. If non-NULL, this overrides the calculation described in <code>rightExpandBy</code> .
<code>rightExpandBy</code>	calculate <code>rightAxisMin</code> and <code>rightAxisMax</code> by multiplicatively extending the data range of the <code>rightAxis</code> series in both directions by this amount. Default value <code>.04</code> extends the data range by 4% in each direction.
<code>rightTicks</code>	number of ticks to draw on right axis at "pretty" locations. Default is 5. This argument is ignored if <code>rightTickLocations</code> is non-NULL.
<code>rightTickLocations</code>	if non-NULL, a vector of desired tick locations or a string that evaluates to such a vector. The default value <code>NULL</code> lets the setting for <code>rightTicks</code> take effect.
<code>labelRightTicks</code>	If <code>TRUE</code> , label ticks on the right axis. Default is <code>FALSE</code> .
<code>rightTickLabels</code>	character vector of tick labels or <code>NULL</code> (the default). If <code>NULL</code> and <code>labelRightTicks</code> is <code>TRUE</code> , labels are constructed from the tick locations. This argument has no effect if <code>labelRightTicks</code> is <code>FALSE</code> .
<code>yCex</code>	<code>cex</code> times this number gives the character expansion factor for the left and right axis labels. Default is 1.
<code>extendTopTick</code>	if <code>TRUE</code> (the default) the top tick of the y axes encloses the panel. <code>leftAxisMax</code> and <code>rightAxisMax</code> are increased as necessary to include the top tick for enclosing the panel.
<code>cex</code>	the base character expansion factor by which all of the <code>***cex</code> parameters are scaled. The default setting is the value of <code>par("cex")</code> .
<code>head</code>	text to appear at the top of the figure region, with alignment determined by <code>headAlign</code> . No default.
<code>headAlign</code>	number indicating justification for the strings in <code>head</code> and <code>sub</code> . 0 means left justify, 1 means right justify, 0.5 (the default) means to center the text. Other numbers are a corresponding distance between the extremes.
<code>headCex</code>	<code>cex</code> times this number gives the character expansion factor for <code>head</code> . Default is 1.5.
<code>sub</code>	text to appear just under <code>head</code> , with alignment determined by <code>headAlign</code> . No default.
<code>subCex</code>	<code>cex</code> times this number gives the character expansion factor for <code>sub</code> . Default is 0.85.
<code>leftTopLabel</code>	text to appear at the top of the left axis, with alignment determined by <code>topLabelAlign</code> . No default.
<code>rightTopLabel</code>	text to appear at the top of the right axis, with alignment determined by <code>topLabelAlign</code> . No default.
<code>topLabelAlign</code>	number indicating alignment for the strings in <code>leftTopLabel</code> . 0 (the default) means left justify, 1 means right justify, 0.5 means to center the text. <code>rightTopLabel</code> , if given, is aligned by <code>1 - topLabelAlign</code> .

labCex	cex times this number gives the character expansion factor for leftTopLabel, rightTopLabel, and xAxisLabel.
leftInsideLabel	text to appear left justified and just inside the upper left corner of the plot region. No default.
rightInsideLabel	text to appear right justified and just inside the upper right corner of the plot region. No default.
innerOffset	number between 0 and 1, a fractional offset for the inside labels. The left edge of leftInsideLabel is offset by this fraction of the x range from the left edge of the plot, as is the right edge of rightInsideLabel from the right edge of the plot.
innerLine	Number of lines in from the top edge of the plot to put the first line of the inside labels.
innerCex	cex times this number gives the character expansion factor for leftInsideLabel and rightInsideLabel.
foot	text to appear at the bottom of the figure region, with alignment determined by footAlign. Use a vector of character strings to get several footnotes. No default.
footAlign	number indicating justification for the strings in foot. 0 (the default) means left justify, 1 means right justify, 0.5 means to center the text. Other numbers are a corresponding distance between the extremes.
footCex	cex times this number gives the character expansion factor for foot. Default is 0.8.
footColor	character or numeric vector as long as foot, specifying the color for each footnote. Default is 1, the device default foreground color. The elements of this argument are cyclically repeated, if necessary, to make footColor the same length as foot.
footSpace	lines of space to set aside directly beneath the space allocated by xSpace to hold footnotes. The space created is footSpace times footCex. Default is length(foot); using a higher value will result in extra space in the bottom figure margin.
tck	The length of xTick, xUnlabeledTick and side tick marks as a fraction of the smaller of the width or height of the plotting region. Minor ticks (xMinorTicks) are 2/3 as long. If tck >= 0.5 it is interpreted as a fraction of the relevant side, so if tck = 1 grid lines are drawn. The default is tck = 0.03.
axisWidth	line width for the axes and box (if any). Default is 2.
start	starting date for the plot. The default is the earliest start time of all the series. This argument can be supplied in any of the forms understood by ti().
end	end date for the plot. The default is the latest end time of all the series. This argument can be supplied in any of the forms understood by ti().
boxType	character representing the type of box. Characters "o", "l" (ell), "7", "c" will produce boxes which resemble the corresponding upper-case letters. The value "n" will suppress boxes. The default is "u".

<code>leftMargin</code>	lines of space for the left margin. Default value (-1) figures this out automatically.
<code>rightMargin</code>	lines of space for the right margin. Default value (-1) figures this out automatically.
<code>topMargin</code>	lines of space for the top margin. Default value (-1) figures this out automatically.
<code>bottomMargin</code>	lines of space for the bottom margin. Default value (-1) figures this out automatically.
<code>nberShade</code>	if TRUE, call the <code>nberShade</code> function to shade recession periods on the plot. The default is FALSE.
<code>nberColor</code>	color to shade recessionary periods if <code>nberShade</code> is TRUE. Passed along to the <code>nberShade</code> function.
<code>nberBorder</code>	if TRUE and <code>nberShade</code> is also TRUE, draws borders on the NBER shaded areas. Default is FALSE.
<code>nberArgs</code>	additional args to be passed along to the <code>nberShade</code> function if <code>nberShade</code> is TRUE.

### Details

`leftAxis`, `plotType`, `lineType`, `lineWidth`, `plotChar`, `dataCex`, `color` and `midPoints` are all cyclically repeated to make them length `nSeries`, the number of series plotted.

Each of the text items `head`, `sub`, `leftTopLabel`, `rightTopLabel`, `leftInsideLabel`, `rightInsideLabel`, `foot`, and `xAxisLabel` can be given as a string, a collections of strings, or as a string that gets evaluated to one of the first two. (But `xAxisLabel` takes only a single string.) See the help details for [evalOrEcho](#) to see how this works.

### Value

`tisPlot` invisibly returns a list of class "tisPlot" and elements named `series`, `dateFormat`, `plotType`, `lineType`, `dataCex`, `color`, `plotChar`, `lineWidth`, `yLegendOffset`, `cex`, `xRange`, `leftRange`, `rightRange`, `midPoints` and `par`. This list is useful mostly as an argument to `tisLegend`.

### Note

The arguments for `tisPlot` and its sister function `scatterPlot` have more descriptive names than the corresponding arguments in `plot`. They are also all of unique types, unlike, for example, the `lty` argument in the usual R plotting functions, which can be either character or numeric. Limiting each argument to a single type was done to make it easier to design a user interface to drive the functions.

Use `tisLegend` to add legends to a plot created by `tisPlot` or `scatterPlot`.

### See Also

[evalOrEcho](#), [scatterPlot](#), [tisLegend](#), [nberShade](#)

## Examples

```
firstTis <- tis(cumsum(rnorm(120)), start = c(1996,1), freq = 12)
secondTis <- tis(cumsum(rnorm(120)), start = c(1996,1), freq = 12)
tisPlot(firstTis, secondTis, color = c("red", "green"),
        lineType = "solid", head = "Two Random Walks")
tisLegend(legend = c("Random Walk 1", "Random Walk 2"))

series <- tis(cumsum(rnorm(200)), start = c(1960,1), tif = "quarterly")
tisPlot(series, xMinorTickFreq = "annual", nberShade = TRUE,
        head = "A Random Walk", sub = "Looks like an econ series",
        rightTopLabel = "$Billions")
romerLines()
```

---

today

*Time Index for the Current Date*

---

## Description

Returns a `ti` for the current date.

## Usage

```
today(tif = "daily")
```

## Arguments

`tif` a `ti` Frequency, given as either a numerical code or a string. `tif()` with no arguments returns a list of the allowable numerical codes and names. The default "daily" returns a `ti` object for the current day.

## Value

A `ti` object of the specified `ti` frequency that contains the current date in the time interval it represents. For example, if `tif` is "monthly", the returned `ti` object will be for the current month.

## See Also

[ti](#), [Sys.Date](#)

---

updateColumns	<i>Update lists and time series</i>
---------------	-------------------------------------

---

### Description

`updateList` compares the names of `oldlist` and `newlist`, deletes the matching elements from a copy of `oldlist`, then returns the result of concatenating that list with `newlist`.

`updateColumns` updates columns of first series from same-named columns of second series using `mergeSeries()`. If second series has columns with names not found in `colnames` of first series, those columns are `cbind()`'ed onto first series.

### Usage

```
updateColumns(oldmat, newmat)
updateList(oldlist, newlist)
```

### Arguments

<code>oldmat</code>	a multivariate <code>tis</code> series
<code>newmat</code>	a multivariate <code>tis</code> series
<code>oldlist</code>	a list
<code>newlist</code>	a list

### Value

`updateList` returns the updated list.

`updateColumns` returns a multivariate `tis` series

### See Also

[tis](#), [mergeSeries](#), [cbind.tis](#)

---

window.tis	<i>Time windows for Time Indexed Series</i>
------------	---

---

### Description

`window.tis` extracts the subset of the object `x` observed between the times `start` and `end`.

### Usage

```
## S3 method for class 'tis':
window(x, start = NULL, end = NULL, extend = FALSE, noWarn = FALSE, ...)
```

**Arguments**

<code>x</code>	a <code>tis</code> object
<code>start</code>	the start time of the period of interest.
<code>end</code>	the end time of the period of interest.
<code>extend</code>	logical. If <code>TRUE</code> , the <code>start</code> and <code>end</code> values are allowed to extend the series. If <code>FALSE</code> , attempts to extend the series are ignored and a warning is issued unless <code>noWarn</code> is <code>FALSE</code> .
<code>noWarn</code>	logical. If <code>FALSE</code> (the default), warnings are generated if <code>extend</code> is <code>FALSE</code> and either (i) <code>start</code> is earlier than the start of the series or (ii) <code>end</code> is later than the end of the series.
<code>...</code>	other arguments to this function are ignored.

**Details**

The start and end times can be `ti` objects, or anything that `ti(z, tif = tif, freq = frequency)`, can turn into a `ti` object.

**Value**

A `tis` object that starts and ends at the given times.

**Note**

The replacement method `window<- .tis` has not been implemented. Use the subscript operator with a `ti` argument to replace values of a `tis` object.

**Examples**

```
z <- tis(1:24, start = c(2001,1), freq = 12)
z2 <- window(z, start = 19991231, extend = TRUE) ## z2 extends back with NA's
window(z, end = end(z) - 3)
```

---

 ymd

---

*Extract parts of various Date-Time Objects*


---

**Description**

Extract the year, month or day, or all three (in `yyyymmdd` form), or the quarter, from a `jul`, `ti`, or from any object that `jul()` can handle.

**Usage**

```
ymd(x, ...)
## S3 method for class 'jul':
ymd(x, ...)
## S3 method for class 'ssDate':
ymd(x, ...)
## S3 method for class 'ti':
ymd(x, offset = 1, ...)
## Default S3 method:
ymd(x, ...)
year(x, ...)
quarter(x, ...)
month(x, ...)
day(x, ...)
```

**Arguments**

<code>x</code>	a <code>ti</code> or <code>jul</code> , or something that <code>jul()</code> can create a <code>jul</code> object from.
<code>...</code>	other args to be passed to the method called by the generic function. <code>year</code> , <code>quarter</code> , <code>month</code> , <code>day</code> and <code>ymd.default</code> may pass these args to <code>as.Date</code> .
<code>offset</code>	for <code>ti</code> <code>x</code> , a number in the range <code>[0,1]</code> telling where in the period represented by <code>x</code> to find the day. <code>0</code> returns the first day of the period, while the default value <code>1</code> returns the last day of the period. For example, if <code>x</code> has <code>tif = "wmonday"</code> so that <code>x</code> represents a week ending on Monday, than any <code>offset</code> in the range <code>[0, 1/7]</code> will return the Tuesday of that week, while <code>offset</code> in the range <code>(1/7, 2/7]</code> will return the Wednesday of that week, <code>offset</code> in the range <code>(6/7, 1]</code> will return the Monday that ends the week, and so on.

**Details**

`year`, `quarter`, `month` and `day` call `ymd`, and thus understand the same arguments as it does. The default implementation `ymd.default` passes it's arguments to a call to the function `jul`, so all of these functions work the same way that function does.

**Value**

`ymd` and it's variants return numeric objects in `yyyymmdd` form.

`year`, `quarter`, `month` and `day` return numeric objects.

`ymd()` with no arguments returns today's `yyyymmdd`.

**See Also**

[jul](#), [ti](#), [as.Date](#)

**Examples**

```
ymd()                                ## today's date and time
weekFromNow <- ymd(today() + 7)      ## today() returns a daily ti
year(jul(today()))
month(Sys.time())
## create a monthly tis (Time Indexed Series)
aTis <- tis(0, start = c(2000, 1), end = c(2004, 12), freq = 12)
ymd(ti(aTis))                        ## the yyyymmdd dates of the observations
```

# Index

- \*Topic **algebra**
  - RowMeans, 49
  - solve.tridiag, 57
- \*Topic **aplot**
  - lines.tis, 41
  - nberShade, 43
  - nberShade.ggplot, 45
  - plotWindow, 46
  - tisLegend, 72
- \*Topic **arith**
  - cumsum.tis, 18
- \*Topic **array**
  - as.matrix.tis, 8
  - RowMeans, 49
  - t.tis, 62
- \*Topic **attribute**
  - description, 23
- \*Topic **character**
  - blanks, 12
  - stripBlanks, 60
- \*Topic **chron**
  - as.Date.jul, 6
  - currentMonday, 19
  - currentPeriod, 20
  - dayOfPeriod, 22
  - format.ti, 25
  - hms, 28
  - holidays, 29
  - inferTi, 31
  - Intraday, 32
  - isIntradayTif, 33
  - isLeapYear, 34
  - jul, 35
  - latestPeriod, 38
  - POSIXct, 47
  - setDefaultFrequencies, 56
  - ssDate, 58
  - ti, 62
  - tiDaily, 65
  - tif, 65
  - tif2freq, 67
  - today, 79
  - ymd, 81
- \*Topic **classes**
  - stripClass, 61
- \*Topic **connection**
  - tisFromCsv, 70
- \*Topic **data**
  - assignList, 9
- \*Topic **dplot**
  - interpNA, 31
- \*Topic **file**
  - csv, 17
  - tisFromCsv, 70
- \*Topic **hplot**
  - scatterPlot, 50
  - screenPage, 54
  - tisPlot, 73
- \*Topic **list**
  - as.list.keepClass, 6
  - as.list.ti, 7
  - columns, 13
  - updateColumns, 80
- \*Topic **manip**
  - as.list.keepClass, 6
  - as.list.ti, 7
  - columns, 13
- \*Topic **math**
  - between, 11
  - linearSplineIntegration, 39
- \*Topic **package**
  - tis-package, 1
- \*Topic **print**
  - csv, 17
- \*Topic **programming**
  - evalOrEcho, 24
- \*Topic **ts**
  - aggregate.tis, 3

- as.data.frame.tis, 5
- as.matrix.tis, 8
- as.ts.tis, 8
- basis, 10
- cbind.tis, 12
- constantGrowthSeries, 14
- convert, 15
- cumsum.tis, 18
- currentMonday, 19
- currentPeriod, 20
- dateRange, 21
- dayOfPeriod, 22
- growth.rate, 27
- interpNA, 31
- lags, 37
- latestPeriod, 38
- lines.tis, 41
- mergeSeries, 42
- naWindow, 42
- print.tis, 48
- RowMeans, 49
- start.tis, 59
- t.tis, 62
- ti, 62
- tiDaily, 65
- tis, 67
- tisFilter, 69
- tisPlot, 73
- today, 79
- updateColumns, 80
- window.tis, 80
- \*Topic utilities**
  - POSIXct, 47
- aggregate, 4, 17, 69
- aggregate.tis, 3
- aggregate.ts (*aggregate.tis*), 3
- apply, 4
- approx, 40
- approxfun, 31, 32
- as.character.jul (*format.ti*), 25
- as.character.ti (*format.ti*), 25
- as.data.frame.tis, 5
- as.Date, 6, 36, 64, 82
- as.Date.jul, 6
- as.Date.ti (*as.Date.jul*), 6
- as.jul (*jul*), 35
- as.list, 7
- as.list.jul (*as.list.ti*), 7
- as.list.keepClass, 6, 7
- as.list.ti, 7
- as.matrix.tis, 8
- as.POSIXct, 48
- as.ssDate (*ssDate*), 58
- as.ti, 31
- as.ti (*ti*), 62
- as.tis (*tis*), 67
- as.ts, 9
- as.ts.tis, 8
- asJul (*jul*), 35
- assign, 9, 10
- assignList, 9
- asTi (*ti*), 62
- basePeriod (*tif*), 65
- basis, 10
- basis<- (*basis*), 10
- between, 11
- blanks, 12, 60
- cbind, 13
- cbind.tis, 12, 42, 69, 80
- class, 61
- columns, 13
- constantGrowthSeries, 14
- convert, 4, 11, 15, 69
- couldBeTi (*ti*), 62
- csv, 17
- cummax, 19
- cummax.tis (*cumsum.tis*), 18
- cummin, 19
- cummin.tis (*cumsum.tis*), 18
- cumprod, 19
- cumprod.tis (*cumsum.tis*), 18
- cumsum, 19
- cumsum.tis, 18
- currentApril (*currentPeriod*), 20
- currentAugust (*currentPeriod*), 20
- currentDecember (*currentPeriod*), 20
- currentFebruary (*currentPeriod*), 20
- currentFriday (*currentMonday*), 19
- currentHalf (*currentPeriod*), 20
- currentJanuary (*currentPeriod*), 20
- currentJuly (*currentPeriod*), 20
- currentJune (*currentPeriod*), 20
- currentMarch (*currentPeriod*), 20

- currentMay (*currentPeriod*), 20
- currentMonday, 19
- currentMonth (*currentPeriod*), 20
- currentMonthDay (*dayOfPeriod*), 22
- currentNovember (*currentPeriod*), 20
- currentOctober (*currentPeriod*), 20
- currentPeriod, 20
- currentQ4 (*currentPeriod*), 20
- currentQMonth (*currentPeriod*), 20
- currentQuarter (*currentPeriod*), 20
- currentSaturday (*currentMonday*), 19
- currentSeptember (*currentPeriod*), 20
- currentSunday (*currentMonday*), 19
- currentThursday (*currentMonday*), 19
- currentTuesday (*currentMonday*), 19
- currentWednesday (*currentMonday*), 19
- currentWeek, 39
- currentWeek (*currentPeriod*), 20
- currentYear (*currentPeriod*), 20
  
- data.frame, 5
- Date, 6
- dateRange, 21
- DateTimeClasses, 48
- day (*ymd*), 81
- dayOfMonth (*dayOfPeriod*), 22
- dayOfPeriod, 22
- dayOfWeek (*dayOfPeriod*), 22
- dayOfYear (*dayOfPeriod*), 22
- description, 23
- description<- (*description*), 23
- documentation (*description*), 23
- documentation<- (*description*), 23
  
- easter (*holidays*), 29
- end, 21, 60
- end.tis, 69
- end.tis (*start.tis*), 59
- environment, 9
- evalOrEcho, 24, 54, 78
  
- fanSeries (*constantGrowthSeries*), 14
- federalHolidays (*holidays*), 29
  
- filter, 70
- firstBusinessDayOf (*dayOfPeriod*), 22
- firstBusinessDayOfMonth (*dayOfPeriod*), 22
- firstDayOf (*dayOfPeriod*), 22
- format.jul (*format.ti*), 25
- format.POSIXlt, 25
- format.ti, 25, 49
- fortify, 26
- fortify.tis, 26
- frequency, 66, 67
  
- geom\_rect, 45, 46
- goodFriday (*holidays*), 29
- growth.rate, 14, 27
- growth.rate<- (*growth.rate*), 27
- gsub, 60
  
- hms, 28
- holidays, 23, 29
- holidaysBetween (*holidays*), 29
- hourly, 29, 34
- hourly (*Intraday*), 32
  
- ilspline (*linearSplineIntegration*), 39
- inaugurationDay (*holidays*), 29
- inferTi, 31
- interpNA, 31
- Intraday, 32
- is.jul (*jul*), 35
- is.ssDate (*ssDate*), 58
- is.ti (*ti*), 62
- is.tis (*tis*), 67
- isEaster (*holidays*), 29
- isGoodFriday (*holidays*), 29
- isHoliday (*holidays*), 29
- isIntradayTif, 33
- isLeapYear, 34
  
- jul, 23, 29, 32, 35, 36, 59, 64, 65, 82
  
- Lag (*lags*), 37
- lag.tis (*lags*), 37
- Lags (*lags*), 37
- lags, 37
- lapply, 4

- lastBusinessDayOf (*dayOfPeriod*),  
22
- lastBusinessDayOfMonth  
(*dayOfPeriod*), 22
- lastDayOf (*dayOfPeriod*), 22
- latestApril (*latestPeriod*), 38
- latestAugust (*latestPeriod*), 38
- latestDecember (*latestPeriod*), 38
- latestFebruary (*latestPeriod*), 38
- latestFriday (*currentMonday*), 19
- latestHalf (*latestPeriod*), 38
- latestJanuary (*latestPeriod*), 38
- latestJuly (*latestPeriod*), 38
- latestJune (*latestPeriod*), 38
- latestMarch (*latestPeriod*), 38
- latestMay (*latestPeriod*), 38
- latestMonday (*currentMonday*), 19
- latestMonth (*latestPeriod*), 38
- latestMonthDay (*dayOfPeriod*), 22
- latestNovember (*latestPeriod*), 38
- latestOctober (*latestPeriod*), 38
- latestPeriod, 38
- latestQ4 (*latestPeriod*), 38
- latestQuarter (*latestPeriod*), 38
- latestSaturday (*currentMonday*), 19
- latestSeptember (*latestPeriod*), 38
- latestSunday (*currentMonday*), 19
- latestThursday (*currentMonday*), 19
- latestTuesday (*currentMonday*), 19
- latestWednesday (*currentMonday*),  
19
- latestWeek, 21
- latestWeek (*latestPeriod*), 38
- latestYear (*latestPeriod*), 38
- legend, 72
- linearSplineIntegration, 39
- lines, 41
- lines.tis, 41, 69, 74
- lintegrate  
(*linearSplineIntegration*),  
39
- mergeSeries, 42, 69, 80
- minutely, 34
- minutely (*Intraday*), 32
- month (*ymd*), 81
- naWindow, 42
- nber.xy (*nberShade*), 43
- nberDates, 46
- nberDates (*nberShade*), 43
- nberShade, 43, 46, 78
- nberShade.ggplot, 44, 45
- nextBusinessDay, 23
- nextBusinessDay (*holidays*), 29
- observed (*basis*), 10
- observed<- (*basis*), 10
- par, 47
- period (*tif*), 65
- plot.window, 47
- plotWindow, 46
- points, 41, 51, 74
- points.tis, 69
- points.tis (*lines.tis*), 41
- polygon, 43, 44
- POSIXct, 47
- POSIXlt (*POSIXct*), 47
- previousBusinessDay, 23
- previousBusinessDay (*holidays*), 29
- print.tis, 48, 69
- print.ts, 49
- quarter (*ymd*), 81
- read.csv, 70, 71
- read.table, 71
- romerLines (*nberShade*), 43
- RowMeans, 49
- rowMeans, 49
- rows (*columns*), 13
- RowSums (*RowMeans*), 49
- rowSums, 49
- scatterPlot, 50, 72, 78
- screenPage, 54
- secondly, 34
- secondly (*Intraday*), 32
- setDefaultFrequencies, 21, 39, 56
- solve, 58
- solve.tridiag, 57
- spline, 40
- splinefun, 31, 32
- split.screen, 56
- ssDate, 58
- start, 21, 60
- start.tis, 59, 69

`start` ← (`start.tis`), 59  
`strftime`, 25, 75  
`stripBlanks`, 60  
`stripClass`, 61  
`stripTis` (`stripClass`), 61  
`strptime`, 70  
`Sys.Date`, 79

`t`, 62  
`t.tis`, 62  
`tapply`, 4  
`ti`, 17, 19, 21, 23, 29, 32, 36, 39, 62, 65, 66,  
69, 71, 79, 82  
`ti.tis`, 69  
`tiBusiness` (`tiDaily`), 65  
`tiDaily`, 65  
`tif`, 17, 21–23, 33, 39, 64, 65, 67  
`tif2freq`, 67  
`tifList` (`setDefaultFrequencies`),  
56  
`tifName`, 33, 57, 64, 67  
`tifName` (`tif`), 65  
time zones, 47  
`tis`, 17, 18, 21, 31, 62, 67, 71, 80  
`tis`-package, 1  
`tisFilter`, 69  
`tisFromCsv`, 70  
`tisLegend`, 72, 78  
`tisPlot`, 54, 72, 73  
`today`, 79  
`try`, 24  
`ts`, 69  
`tunnelSeries`  
    (`constantGrowthSeries`), 14

`updateColumns`, 80  
`updateList` (`updateColumns`), 80

`window`, 43  
`window.tis`, 69, 80  
`write.table`, 18

`year` (`ymd`), 81  
`ymd`, 36, 64, 81