quantmod

June 9, 2008

Type Package

Title Quantitative Financial Modelling Framework

Version 0.3-6

Revision 433

Date 2008-06-09

Author Jeffrey A. Ryan

Depends xts(>= 0.0-15),zoo,Defaults

Suggests DBI,RMySQL,RSQLite,TTR(>= 0.14-0),fSeries,its

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

Description Specify, build, trade, and analyse quantitative financial trading strategies

LazyLoad yes

License GPL-3

URL http://www.quantmod.com http://r-forge.r-project.org/projects/quantmod

R topics documented:

Delt ................................................................. 2
Lag ................................................................ 3
Next ................................................................. 5
OHLCTransformations ............................................. 6
TA ................................................................. 8
addADX ............................................................. 10
addBBands ......................................................... 11
addExpiry .......................................................... 12
addMA ............................................................. 12
addMACD ......................................................... 14
addROC ............................................................ 15
addRSI ............................................................. 16
addSAR ............................................................ 17
addSMI ............................................................ 18
addVo .............................................................. 19
### Delt

**Description**

Calculate the k-period percent difference within one series, or between two series. Primarily used to calculate the percent change from one period to another of a given series, or to calculate the percent difference between two series over the full series.
Usage

Delt(x1, x2 = NULL, k = 0, type = c("arithmetic", "log"))

Arguments

- **x1**: A m x 1 vector
- **x2**: A m x 1 vector (optional)
- **k**: Change over k-periods. Default k=1 when x2 is NULL.
- **type**: Type of difference. "log" or "arithmetic" (default).

Details

When called with only x1, the one period percent change of the series is returned by default. Internally, this happens by copying x1 to x2. A two period difference would be specified with k=2.

If called with both x1 and x2, the difference between the two is returned. That is, k=0. A one period difference would be specified by k=1. k may also be a vector to calculate more than one period at a time. The results will then be an m x length(k).

Log differences are used by default: $\text{Lag} = \log(x2(t)/x1(t-k))$

Arithmetic differences are calculated: $\text{Lag} = (x2(t) - x1(t-k))/x1(t-k)$

Value

An m x length(x1) matrix of rows and length(k) columns.

Author(s)

Jeffrey A. Ryan

See Also

- OpOp
- OpCl

Examples

```r
Stock.Open <- c(102.25, 102.87, 102.25, 100.87, 103.44, 103.87, 103.00)
Stock.Close <- c(102.12, 102.62, 100.12, 103.00, 103.87, 103.12, 105.12)

delt(Stock.Open)  # one period pct. price change
Delt(Stock.Open,k=1)  # same
Delt(Stock.Open,type='arithmetic')  # using arithmetic differences
Delt(Stock.Open,Stock.Close)  # Open to Close pct. change
Delt(Stock.Open,Stock.Close,k=0:2)  # ... for 0,1, and 2 periods
```
Description

Create a lagged series from data, with NA used to fill.

Usage

Lag(x, k = 1)

## S3 method for class 'quantmod.OHLC':
Lag(x, k = 1)

## S3 method for class 'zoo':
Lag(x, k = 1)

## S3 method for class 'data.frame':
Lag(x, k = 1)

## S3 method for class 'numeric':
Lag(x, k = 1)

Arguments

x    vector or series to be lagged
k    periods to lag.

Details

Shift series k-periods down, prepending NAs to front of series.
Specifically designed to handle quantmod.OHLC and zoo series within the quantmod workflow.
If no S3 method is found, a call to lag in base is made.

Value

The original x prepended with k NAs and missing the trailing k values.
The returned series maintains the number of obs. of the original.

Note

This function differs from lag by returning the original series modified, as opposed to simply changing the time series properties. It differs from the like named Lag in the Hmisc as it deals primarily with time-series like objects.
It is important to realize that if there is no applicable method for Lag, the value returned will be from lag in base. That is, coerced to ‘ts’ if necessary, and subsequently shifted.

Author(s)

Jeffrey A. Ryan
Next

See Also

lag

Examples

```r
Stock.Close <- c(102.12,102.62,100.12,103.00,103.87,103.12,105.12)
Close.Dates <- as.Date(c(10660,10661,10662,10665,10666,10667,10668),origin="1970-01-01")

Lag(Stock.Close)  #lag by 1 period
Lag(Stock.Close,k=1)  #same
Lag(Stock.Close,k=1:3)  #lag 1,2 and 3 periods
```

Description

Create a new series with all values advanced forward one period. The value of period 1, becomes the value at period 2, value at 2 becomes the original value at 3, etc. The opposite of Lag. NA is used to fill.

Usage

```r
Next(x, k = 1)
```

## S3 method for class 'quantmod.OHLC':
```r
Next(x,k=1)
```

## S3 method for class 'zoo':
```r
Next(x,k=1)
```

## S3 method for class 'data.frame':
```r
Next(x,k=1)
```

## S3 method for class 'numeric':
```r
Next(x,k=1)
```

Arguments

- `x` vector or series to be advanced
- `k` periods to advance

Details

Shift series k-periods up, appending NAs to end of series.

Specifically designed to handle quantmod.OHLC and zoo series within the quantmod workflow.

If no S3 method is found, a call to lag in base is made, with the indexing reversed to shift the time series forward.
Value

The original \( x \) appended with \( k \) NAs and missing the leading \( k \) values.

The returned series maintains the number of obs. of the original.

Unlike \texttt{Lag}, only one value for \( k \) is allowed.

Note

This function’s purpose is to get the “next” value of the data you hope to forecast, e.g. a stock’s closing value at \( t+1 \). Specifically to be used within the \texttt{quantmod} framework of \texttt{specifyModel}, as a functional wrapper to the LHS of the model equation.

It is not magic - and thus will not get tomorrow’s values...

Author(s)

Jeffrey A. Ryan

See Also

\texttt{specifyModel}, \texttt{Lag}

Examples

```r
Stock.Close <- c(102.12,102.62,100.12,103.00,103.87,103.12,105.12)
Close.Dates <- as.Date(c(10660,10661,10662,10665,10666,10667,10668),origin="1970-01-01")

Next(Stock.Close) #one period ahead
Next(Stock.Close,k=1) #same

merge(Next(Stock.Close),Stock.Close)
```

## Not run:

# a simple way to build a model of next days
# IBM close, given todays. Technically both
# methods are equal, though the former is seen
# as more intuitive...ymmv

```r
specifyModel(Next(Cl(IBM)) ~ Cl(IBM))
specifyModel(Cl(IBM) ~ Lag(Cl(IBM)))
```

## End(Not run)
In the case of functions consisting of combined Op, Hi, Lo, Cl (e.g. ClCl(x)) the one period transformation will be applied. For example, to return the Open to Close of an object it is possible to call OpCl(x). If multiple periods are desired a call to the function Delt is necessary. 

seriesLo and seriesHi will return the low and high, respectively, of a given series.
HLC extracts the High, Low, and Close columns. OHLC extracts the Open, High, Low, and Close columns.

These functions are merely to speed the model specification process. All columns may also be extracted through standard R methods.
Assignment will not work at present.

Usage

Op(x)  
Hi(x)  
Lo(x)  
Cl(x)  
Vo(x)  
Ad(x)  
seriesHi(x)  
seriesLo(x)  

OpCl(x)  
ClCl(x)  
HiCl(x)  
LoCl(x)  
LoHi(x)  
OpHi(x)  
OpLo(x)  
OpOp(x)  

HLC(x)  
OHLC(x)  

Arguments

x A data object with columns containing data to be extracted.

Details

Internally, the code uses grep to locate the appropriate columns. Therefore it is necessary to use inputs with column names matching the requirements in the description section, though the exact naming convention is not as important.

Value

Returns an object of the same class as the original series, with the appropriately column names if applicable and/or possible. The only exceptions are for quantmod.OHLC objects which will be returned as zoo objects, and calls to seriesLo and seriesHi which may return a numeric value instead of the original object type.
Author(s)

Jeffrey A. Ryan

See Also

specifyModel

Examples

```r
## Not run:
getSymbols('IBM', src='yahoo')
Ad(IBM)
Cl(IBM)
ClCl(IBM)

seriesHi(IBM)
seriesHi(Lo(IBM))

removeSymbols('IBM')
## End(Not run)
```

---

Add Technical Indicator to Chart

Description

Functions to add technical indicators to a chart.

Details

The general mechanism to add technical analysis studies or overlays to a financial chart created with `chartSeries`.

Functionality marked with a `*` is via the TTR package.

General TA charting tool functions:

- `addTA` add data as custom indicator
- `dropTA` remove technical indicator
- `moveTA` move a technical indicator
- `swapTA` swap two technical indicators

Current technical indicators include:

- `addADX` add Welles Wilder’s Directional Movement Indicator*
- `addATR` add Average True Range *
- `addBBands` add Bollinger Bands *
- `addCCI` add Commodity Channel Index *
- `addCMF` add Chaiken Money Flow *
- `addCMO` add Chande Momentum Oscillator *
- `addDEMA` add Double Exponential Moving Average *
addDPO  add Detrended Price Oscillator *
addEMA  add Exponential Moving Average *
addEnvelope  add Moving Average Envelope
addEVWMA  add Exponential Volume Weighted Moving Average *
addExpiry  add options or futures expiration lines
addLines  add line(s)
addMACD:  add Moving Average Convergence Divergence *
addMomentum  add Momentum *
addPoints  add point(s)
addROC:  add Rate of Change *
addRSI  add Relative Strength Indicator *
addSAR  add Parabolic SAR *
addSMA  add Simple Moving Average *
addSMI  add Stochastic Momentum Index *
addTRIX  add Triple Smoothed Exponential Oscillator *
addVo:  add Volume if available
addWMA  add Weighted Moving Average *
addWPR  add Williams Percent R *
addZLEMA  add ZLEMA *

See the individual functions for specific implementation and argument details. Details of the underlying TTR implementations can be found in TTR.

The primary changes between the add*** version of an indicator and the TTR base function is the absence of the data argument in the former.

Notable additions include on, with.col and overlay (deprecated).

Value

Called for its side effects, an object to class chobTA will be returned invisibly. If called from the R command line the method will draw the appropriate indicator on the current chart.

Note

Calling any of the above methods from within a function or script will generally require them to be wrapped in a plot call as they rely on the context of the call to initiate the actual charting addition.

Author(s)

Jeffrey A. Ryan

References

Josh Ulrich - TTR package
addADX  

Add Directional Movement Index

Description
Add Directional Movement Index

Usage
addADX(n = 14, maType="EMA", wilder=TRUE)

Arguments
n periods to use for DX calculation
maType moving average type
wilder should Welles Wilder EMA be used?

Details
See ’ADX’ in TTR for specific details and references.

Value
An ADX indicator will be draw in a new window on the current chart. A chobTA object will be returned silently.

Author(s)
Jeffrey A. Ryan

References
see ADX in TTR written by Josh Ulrich

See Also
addTA

Examples
## Not run:
addADX()
## End(Not run)
addBBands  Add Bollinger Bands to Chart

Description

Add Bollinger Bands to current chart.

Usage

addBBands(n = 20, sd = 2, ma = "SMA", draw = 'bands', on = -1)

Arguments

- **n**: number of moving average periods
- **ma**: type of moving average to be used
- **sd**: number of standard deviations
- **draw**: indicator to draw: bands, percent, or width
- **on**: which figure area of chart to apply to

Details

The primary addition to this function call over the TTR version is in the `draw` argument. 'bands' will draw standard Bollinger Bands, 'percent' will draw Bollinger %b and 'width' will draw Bollinger Bands Width. The last two will be drawn in new figure regions.

See bollingerBands in TTR for specific details as to implementation and references.

Value

Bollinger Bands will be drawn, or scheduled to be drawn, on the current chart. If `draw` is either percent or width a new figure will be added to the current TA figures charted.

A chobTA object will be returned silently.

Author(s)

Jeffrey A. Ryan

References

See bollingerBands in TTR written by Josh Ulrich

See Also

addTA

Examples

```r
## Not run:
addBBands()
## End(Not run)
```
addExpirey  

*Add Contract Expiration Bars to Chart*

**Description**

Apply options or futures expiration vertical bars to current chart.

**Usage**

```r
addExpirey(type = "options", lty = "dotted")
```

**Arguments**

- `type`: options or futures expiration
- `lty`: type of lines to draw

**Details**

See options.expiry and futures.expiry in `quantmod` for details and limitations.

**Value**

Expiration lines will be drawn at appropriate dates. A chibTA object will be returned silently.

**Author(s)**

Jeffrey A. Ryan

**See Also**

`addTA`

**Examples**

```r
## Not run:
addExpirey()
## End(Not run)
```

---

addMA  

*Add Moving Average to Chart*

**Description**

Add one or more moving averages to a chart.
addMA

Usage

addSMA(n = 10, on = 1, with.col = Cl, overlay = TRUE, col = "brown")
addEMA(n = 10, wilder = FALSE, ratio=NULL, on = 1,
       with.col = Cl, overlay = TRUE, col = "blue")
addWMA(n = 10, wts=1:n, on = 1, with.col = Cl, overlay = TRUE, col = "green")
addDEMA(n = 10, on = 1, with.col = Cl, overlay = TRUE, col = "pink")
addEVWMA(n = 10, on = 1, with.col = Cl, overlay = TRUE, col = "yellow")
addZLEMA(n = 10, ratio=NULL, on = 1, with.col = Cl, overlay = TRUE, col = "red")

Arguments

n  periods to average over
wilder  logical; use wilder?
wts  a vector of weights
ratio  a smoothing/decay ratio
on  apply to which figure (see below)
with.col  using which column of data (see below)
overlay  draw as overlay
col  color of MA

Details

see the appropriate base MA functions in TTR for more details and references.

Value

A moving average indicator will be draw on the current chart. A chobTA object will be returned silently.

Author(s)

Jeffrey A. Ryan

References

see MovingAverages in pkgTTR written by Josh Ulrich

See Also

addTA
addMACD

Example

```r
## Not run:
addSMA()
addEMA()
addWMA()
addDEMA()
addEVWMA()
addZLEMA()
## End(Not run)
```

---

**Description**

Add Moving Average Convergence Divergence indicator to chart.

**Usage**

```r
addMACD(fast = 12, slow = 26, signal = 9, type = "EMA", histogram = TRUE, col)
```

**Arguments**

- `fast`: fast period
- `slow`: slow period
- `signal`: signal period
- `type`: type of MA to use. Single values will be replicated
- `histogram`: include histogram
- `col`: colors to use for lines (optional)

**Details**

See and 'MACD' in TTR for specific details and implementation references.

**Value**

A MACD indicator will be draw in a new window on the current chart. A chobTA object will be returned silently.

**Author(s)**

Jeffrey A. Ryan

**References**

see MACD in TTR written by Josh Ulrich

**See Also**

addTA
addROC

Add Rate Of Change to Chart

Description
Add Rate Of Change indicator to chart.

Usage
addROC(n = 1, type = c("discrete", "continuous"), col = "red")

Arguments
- n: periods
- type: compounding type
- col: line color (optional)

Details
See "ROC" in TTR for specific details and references.

Value
A ROC indicator will be draw in a new window on the current chart. A chobTA object will be returned silently.

Author(s)
Jeffrey A. Ryan

References
see ROC in TTR written by Josh Ulrich

See Also
addTA

Examples
## Not run:
addMACD()
## End(Not run)
addRSI

Add Relative Strength Index to Chart

Description

Add a Relative Strength Index indicator to chart.

Usage

addRSI(n = 14, type = "EMA", wilder = TRUE)

Arguments

n  periods

type  type of MA to use

wilder  use wilder (see EMA)

Details

see 'RSI' in TTR for specific details and references.

Value

An RSI indicator will be draw in a new window on the current chart. A chobTA object will be returned silently.

Author(s)

Jeffrey A. Ryan

References

see RSI in TTR written by Josh Ulrich

See Also

addTA

Examples

## Not run:
addRSI()
## End(Not run)
addSAR

Add Parabolic Stop and Reversal to Chart

Description

Add Parabolic Stop and Reversal indicator overlay to chart.

Usage

addSAR(accel = c(0.02, 0.2), col = "blue")

Arguments

accel Accelleration factors - see SAR
col color of points (optional)

Details

see 'SAR' in TTR for specific details and references.

Value

A SAR overlay will be drawn on the current chart. A chobTA object will be returned silently.

Author(s)

Jeffrey A. Ryan

References

see SAR in TTR written by Josh Ulrich

See Also

addTA

Examples

## Not run:
addSAR()
## End(Not run)
**addSMI**

Add Stochastic Momentum Indicator to Chart

**Description**

Add Stochastic Momentum Indicator to chart.

**Usage**

```r
addSMI(n=13, slow=25, fast=2, signal=9, ma.type="EMA")
```

**Arguments**

- `n`: periods
- `slow`: slow
- `fast`: fast
- `signal`: signal
- `ma.type`: MA type to use, recycled as necessary

**Details**

see `SMI` in `TTR` for specifics and references.

**Value**

An SMI indicator will be draw in a new window on the current chart. A chobTA object will be returned silently.

**Author(s)**

Jeffrey A. Ryan

**References**

see SMI in `TTR` written by Josh Ulrich

**See Also**

`addTA`

**Examples**

```r
## Not run:
addSMI()
## End(Not run)
```
**addVo**  

Add Volume to Chart

**Description**
Add Volume of a series, if available, to the current chart. This is the default TA argument for all charting functions.

**Usage**

```r
addVo()
```

**Details**
Add volume bars to current chart if data object contains appropriate volume column.

**Value**
Volume will be draw in a new window on the current chart. A chobTA object will be returned silently.

**Author(s)**
Jeffrey A. Ryan

**See Also**

addTA

**Examples**

```r
## Not run:
addVo()
## End(Not run)
```

---

**addWPR**  

Add William’s Percent R to Chart

**Description**
Add William’s percent R indicator to the current chart.

**Usage**

```r
addWPR(n = 14)
```

**Arguments**

- **n**  
  periods

---
buildData

Details

see 'WPR' in TTR for details and references.

Value

A William’s percent R indicator will be draw in a new window on the current chart. A chobTA object will be returned silently.

Author(s)

Jeffrey A. Ryan

References

see 'WPR' in TTR written by Josh Ulrich

See Also

addTA

Examples

## Not run:
addWPR()
## End(Not run)

buildData Create Data Object for Modelling

Description

Create one data object from multiple sources, applying transformations via standard \textit{R} formula mechanism.

Usage

buildData(formula, na.rm = TRUE, return.class = "zoo")

Arguments

\begin{itemize}
\item \texttt{formula} an object of class \texttt{formula} (or one that can be coerced to that class): a symbolic description of the desired output data object, with the dependent side corresponding to first column, and the independent parameters of the formula assigned to the remaining columns.
\item \texttt{na.rm} drop rows with missing values?
\item \texttt{return.class} one of "zoo","data.frame","ts","its","timeSeries"
\end{itemize}
buildModel

Details

Makes available for use outside the quantmod workflow a dataset of appropriately transformed variables, using the same mechanism underlying specifyModel. Offers the ability to apply transformations to raw data using a common formula mechanism, without having to explicitly merge different data objects.

Interally calls specifyModel followed by modelData, with the returned object being coerced to the desired 'return.class' if possible, otherwise returns a zoo object.

See getSymbols and specifyModel for more information regarding proper usage.

Value

An object of class return.class.

Author(s)

Jeffrey A. Ryan

See Also

getSymbols, specifyModel, modelData

Examples

## Not run:
  buildData(Next (OpCl(DIA)) ~ Lag(TBILL) + I(Lag(OpHi(DIA))^2))
  buildData(Next (OpCl(DIA)) ~ Lag(TBILL), na.rm=FALSE)
  buildData(Next (OpCl(DIA)) ~ Lag(TBILL), na.rm=FALSE, return.class="ts")

## End(Not run)

buildModel  

Build quantmod model given specified fitting method

Description

Construct and attach a fitted model of type method to quantmod object.

Usage

buildModel(x, method, training.per, ...)

Arguments

x  
An object of class quantmod created with specifyModel or an R formula

training.per  
character vector representing dates in ISO 8601 format “CCYY-MM-DD” or “CCYY-MM-DD HH:MM:SS” of length 2

method  
A character string naming the fitting method. See details section for available methods, and how to create new methods.

...  
Additional arguments to method call
Details

Currently available methods include:

- lm, glm, loess, step, ppr, rpart[part], tree[tree], randomForest[randomForest], mars[mda], poly-
mars[polspine], lars[lars], rq[quantreg], lqs[MASS], rlm[MASS], svm[e1071], and nnet[nnet].

The `training.per` should match the underlying date format of the time-series data used in mod-
elling. Any other style may not return what you expect.

Additional methods wrappers can be created to allow for modelling using custom functions. The
only requirements are for a wrapper function to be constructed taking parameters `quantmod`,
`training.data`, and .... The function must return the fitted model object and have a pre-
dict method available. It is possible to add predict methods if non exist by adding an S3 method for
predictModel. The `buildModel.skeleton` function can be used for new methods.

Value

An object of class `quantmod` with fitted model attached

Note

See `buildModel.skeleton` for information on adding additional methods

Author(s)

Jeffrey Ryan

See Also

- `specifyModel`
- `tradeModel`

Examples

```r
## Not run:
getSymbols('QQQQ', src='yahoo')
q.model = specifyModel(Next(OpCl(QQQQ)) ~ Lag(OpHi(QQQQ),0:3))
buildModel(q.model, method='lm', training.per=c('2006-08-01','2006-09-30'))
## End(Not run)
```

chartSeries

Create Financial Charts

Description

Charting tool to create standard financial charts given a time series like object. Serves as the base
function for future technical analysis additions. Possible chart styles include candles, matches (1
pixel candles), bars, and lines. Chart may have white or black background.

reChart allows for dynamic changes to the chart without having to respecify the full chart pa-
rameters.
Usage

chartSeries(x,
  type = c("auto", "candlesticks", "matchsticks", "bars","line"),
  subset = NULL,
  show.grid = TRUE,
  name = NULL,
  time.scale = NULL,
  TA = 'addVo()',
  TAs = ';',
  line.type = "l",
  bar.type = "ohlc",
  theme = chartTheme("black"),
  layout = NA,
  major.ticks = 'auto', minor.ticks = TRUE,
  up.col, dn.col, color.vol = TRUE, multi.col = FALSE,
  ...
)

reChart(type = c("auto", "candlesticks", "matchsticks", "bars","line"),
  subset = NULL,
  show.grid = TRUE,
  name = NULL,
  time.scale = NULL,
  line.type = "l",
  bar.type = "ohlc",
  theme = chartTheme("black"),
  major.ticks = 'auto', minor.ticks = TRUE,
  up.col, dn.col, color.vol = TRUE, multi.col = FALSE,
  ...)

Arguments

x an OHLC object - see details

xtype style of chart to draw

subset xts style date subsetting argument

show.grid display price grid lines?

name name of chart

xtime.scale what is the timescale? automatically deduced

xTA a vector of technical indicators and params, or character strings

xTAsenxTA delimiter for TA strings

xline.type type of line in line chart

xbar.type type of barchart - ohlc or hlc

xtheme a chart.theme object

xlayout if NULL bypass internal layout

xmajorticks where should major ticks be drawn

xminorticks should minor ticks be draw?

xup.col up bar/candle color

xdn.col down bar/candle color
Details

Currently displays standard style OHLC charts familiar in financial applications, or line charts when not passes OHLC data. Works with objects having explicit time-series properties.

Line charts are created with close data, or from single column time series.

The subset argument can be used to specify a particular area of the series to view. The underlying series is left intact to allow for TA functions to use the full data set. Additionally, it is possible to use syntax borrowed from the first and last functions, e.g. ‘last 4 months’.

TA allows for the inclusion of a variety of chart overlays and technical indicators. A full list is available from addTA. The default TA argument is addVo() - which adds volume, if available, to the chart being drawn.

theme requires an object of class chart.theme, created by a call to chartTheme. This function can be used to modify the look of the resulting chart. See chart.theme for details.

line.type and bar.type allow further fine tuning of chart styles to user tastes.

multi.col implements a color coding scheme used in some charting applications, and follows the following rules:

grey => Op[t] < Cl[t] and Op[t] < Cl[t-1]
white => Op[t] < Cl[t] and Op[t] > Cl[t-1]
red => Op[t] > Cl[t] and Op[t] < Cl[t-1]
black => Op[t] > Cl[t] and Op[t] > Cl[t-1]

reChart takes any number of arguments from the original chart call — and redraws the chart with the updated parameters. One item of note: if multiple color bars/candles are desired, it is necessary to respecify the theme argument. Additionally it is not possible to change TA parameters at present. This must be done with addTA/dropTA/swapTA/moveTA commands.

Value

Returns a standard chart plus volume, if available, suitably scaled.

Note

Most details can be fine-tuned within the function, though the code does a reasonable job of scaling and labelling axes for the user.

The current implementation maintains a record of actions carried out for any particular chart. This is used to recreate the original when adding new indicator. A list of applied TA actions is available with a call to listTA. This list can be assigned to a variable and used in new chart calls to recreate a set of technical indicators. It is also possible to force all future charts to use the same indicators by calling setTA.

Additional motivation to add outlined candles to allow for scaling and advanced color coding is owed to Josh Ulrich, as are the base functions (from TTR) for the yet to be released technical analysis charting code.

Many improvements in the current version were the result of conversations with Gabor Grothendieck. Many thanks to him.
chartTheme

Author(s)
Jeffrey A. Ryan

References
Josh Ulrich - TTR package and multi.col coding

See Also
getsymbols, addta, setTA, chartTheme

Examples
```r
## Not run:
getsymbols("YHOO")
chartSeries(YHOO)
chartSeries(YHOO, subset='last 4 months')
chartSeries(YHOO, subset='2007::2008-01')
chartSeries(YHOO, theme=chartTheme('white'))
chartSeries(YHOO,TA=NULL)   #no volume
chartSeries(YHOO,TA=c(addVo(),addBBands()))  #add volume and Bollinger Bands from TTR
addMACD()       # add MACD indicator to current chart
setTA()
chartSeries(YHOO)  #draws chart again, this time will all indicators present
## End(Not run)
```

chartTheme  
Create A Chart Theme

Description
Create a chart.theme object for use within chartSeries to manage desired chart colors.

Usage
```r
chartTheme(theme = "black", ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theme</td>
<td>name of base theme</td>
</tr>
<tr>
<td>...</td>
<td>name=value pairs to modify</td>
</tr>
</tbody>
</table>

Details
Used as an argument to the chartSeries family of functions, chartTheme allows for on-the-fly modification of pre-specified chart 'themes'. Users can modify a pre-built theme in-place, or copy the theme to a new variable for use in subsequent charting calls.

Internally a chart.theme object is nothing more than a list of values organized by chart components. The primary purpose of this is to facilitate minor modification on the fly, as well as provide a template for larger changes.
Setting style arguments for TA calls via chartTheme requires the user to pass the styles as name=value pairs with a name containing the TA call in question. See examples for assistance.

Current components that may be modified with appropriate values:

- `fg.col` foreground color
- `bg.col` background color
- `grid.col` grid color
- `border` border color
- `minor.tick` minor tickmark color
- `major.tick` major tickmark color
- `up.col` up bar/candle color
- `dn.col` down bar/candle color
- `up.up.col` up after up bar/candle color
- `up.dn.col` up after down bar/candle color
- `dn.dn.col` down after down bar/candle color
- `dn.up.col` down after up bar/candle color
- `up.border` up bar/candle border color
- `dn.border` down bar/candle border color
- `up.up.border` up after up bar/candle border color
- `up.dn.border` up after down bar/candle border color
- `dn.dn.border` down after down bar/candle border color
- `dn.up.border` down after up bar/candle border color
- `up.up.border` up after up bar/candle border color

**Value**

A chart.theme object

**Author(s)**

Jeffrey A. Ryan

**See Also**

- chartSeries

**Examples**

```r
chartTheme()
chartTheme('white')
chartTheme('white', up.col='blue', dn.col='red')

# A TA example
chartTheme(addRSI.col='red')

str(chartTheme())
```
A Chart Object Class

Description
Internal Objects for Tracking and Plotting Chart Changes

Objects from the Class
Objects are created internally through the charting functions chartSeries, barChart, lineChart, and candleChart.

Slots

- device: Object of class "ANY"
- call: Object of class "call"
- name: Object of class "character"
- type: Object of class "character"
- passed.args: Object of class "ANY"
- windows: Object of class "numeric"
- xrange: Object of class "numeric"
- yrange: Object of class "numeric"
- length: Object of class "numeric"
- color.vol: Object of class "logical"
- multi.col: Object of class "logical"
- show.vol: Object of class "logical"
- show.grid: Object of class "logical"
- line.type: Object of class "character"
- bar.type: Object of class "character"
- xlab: Object of class "character"
- ylab: Object of class "character"
- spacing: Object of class "numeric"
- width: Object of class "numeric"
- bp: Object of class "numeric"
- x.labels: Object of class "character"
- colors: Object of class "ANY"
- time.scale: Object of class "ANY"
- major.ticks: Object of class "ANY"
- minor.ticks: Object of class "logical"

Methods
No methods defined with class "chob" in the signature.
Author(s)
Jeffrey A. Ryan

See Also
chartSeries, or chobTA for links to other classes

Examples
showClass("chob")

---

chobTA-class A Technical Analysis Chart Object

Description
Internal storage class for handling arbitrary TA objects

Objects from the Class
Objects of class chobTA are created and returned invisibly through calls to addTA-style functions.

Slots
- **call**: Object of class "call"
- **on**: Object of class "ANY"
- **new**: Object of class "logical"
- **TA.values**: Object of class "ANY"
- **name**: Object of class "character"
- **params**: Object of class "ANY"

Methods
- **show** signature(object = "chobTA"):

Note
It is of no external value to handle chobTA objects directly

Author(s)
Jeffrey A. Ryan

See Also
addTA, or chob for links to other classes

Examples
showClass("chobTA")
fittedModel

quantmod Fitted Objects

Description

Extract and replace fitted models from quantmod objects built with buildModel. All objects fitted through methods specified in buildModel calls can be extracted for further analysis.

Usage

fittedModel(object)
fittedModel(object) <- value

## S3 method for class 'quantmod':
formula(x, ...)

## S3 method for class 'quantmod':
plot(x, ...)

## S3 method for class 'quantmod':
coefficients(object, ...)

## S3 method for class 'quantmod':
coef(object, ...)

## S3 method for class 'quantmod':
residuals(object, ...)

## S3 method for class 'quantmod':
resid(object, ...)

## S3 method for class 'quantmod':
fitted.values(object, ...)

## S3 method for class 'quantmod':
fitted(object, ...)

## S3 method for class 'quantmod':
anova(object, ...)

## S3 method for class 'quantmod':
logLik(object, ...)

## S3 method for class 'quantmod':
vcov(object, ...)

Arguments

object  a quantmod object
x       a quantmod object
value: a new fitted model

... additional arguments

Details

Most often used to extract the final fitted object of the modelling process, usually for further analysis with tools outside the `quantmod` package.

Most common methods to apply to fitted objects are available to the parent `quantmod` object. At present, one can call directly the following S3 methods on a built model as if calling directly on the fitted object. See *Usage* section.

It is also possible to add a fitted model to an object. This may be of value when applying heuristic rule sets for trading approaches, or when fine tuning already fit models by hand.

Value

Returns an object matching that returned by a call to the method specified in `buildModel`.

Note

The replacement function `fittedModel<-` is highly experimental, and may or may not continue into further releases.

The fitted model added *must* use the same names as appear in the `quantmod` object’s dataset.

Author(s)

Jeffrey A. Ryan

See Also

`quantmod`, `buildModel`

Examples

```r
## Not run:
x <- specifyModel(Next(OpCl(DIA)) ~ OpCl(VIX))
x.lm <- buildModel(x, method='lm', training.per=c('2001-01-01', '2001-04-01'))
fittedModel(x.lm)
coef(fittedModel(x.lm))
coef(x.lm)  # same
vcov(fittedModel(x.lm))
vcov(x.lm)  # same
## End(Not run)
```
getDividends

Description
Download, or download and append stock dividend data from Yahoo! Finance.

Usage
getDividends(Symbol, from = "1970-01-01", to = Sys.Date(), env = .GlobalEnv, src = "yahoo", auto.assign = TRUE, auto.update = TRUE, verbose = FALSE, ...)

Arguments
Symbol          The Yahoo! stock symbol
from            date from in CCYY-MM-DD format
to              date to in CCYY-MM-DD format
evac            where to create object
src              data source (only yahoo is valid at present)
auto.assign     should results be loaded to env
auto.update     automatically add dividend to data object
verbose         display status of retrieval
...              currently unused

Details
Eventually destined to be a wrapper function along the lines of getSymbols to different sources - this currently only support Yahoo data.

Value
If auto.assign is TRUE, the symbol will be written to the environment specified in env with a .div appended to the name.
If auto.update is TRUE and the object is of class xts, the dividends will be included as an attribute of the original object and be reassigned to the environment specified by env.
All other cases will return the dividend data as an xts object.

Note
This function is very preliminary - and will most likely change significantly in the future.
Author(s)
Jeffrey A. Ryan

References
Yahoo! Finance: http://finance.yahoo.com

See Also
getSymbols

Examples

```r
## Not run:
getSymbols("MSFT")
getDividends("MSFT")

getDividends(MSFT)
## End(Not run)
```

description
Download exchange rates or metals prices from oanda.

Usage

```r
getFX(Currencies, 
    from = "2007-01-01", to = Sys.Date(), 
    env = .GlobalEnv, 
    verbose = FALSE, 
    warning = TRUE, 
    auto.assign = TRUE, ...) 
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currencies</td>
<td>Currency pairs expressed as ‘CUR/CUR’</td>
</tr>
<tr>
<td>from</td>
<td>start date expressed in ISO CCYY-MM-DD format</td>
</tr>
<tr>
<td>to</td>
<td>end date expressed in ISO CCYY-MM-DD format</td>
</tr>
<tr>
<td>env</td>
<td>which environment should they be loaded into</td>
</tr>
<tr>
<td>verbose</td>
<td>be verbose</td>
</tr>
<tr>
<td>warning</td>
<td>show warnings</td>
</tr>
<tr>
<td>auto.assign</td>
<td>use auto.assign</td>
</tr>
</tbody>
</table>

Details

A convenience wrapper to getSymbols(x, src='oanda'). See getSymbols and getSymbols.oanda for more detail.
Value

The results of the call will be the data will be assigned automatically to the environment specified (global by default). Additionally a vector of downloaded symbol names will be returned. See getSymbols and getSymbols.oanda for more detail.

Author(s)

Jeffrey A. Ryan

References

Oanda.com http://www.oanda.com

See Also

getSymbols, getSymbols.oanda

Examples

## Not run:
getFX("USD/JPY")
getFX("EUR/USD", from="2005-01-01")
## End(Not run)

getFinancials  Download and View Financial Statements

Description


Usage

getFinancials(Symbol, env = .GlobalEnv, src = "google",
auto.assign = TRUE,
...)

viewFinancials(x, type=c('BS','IS','CF'), period=c('A','Q'),
subset = NULL)

Arguments

Symbol a valid google symbol
env where to create the object
src currently unused
auto.assign should results be loaded to the environment
... currently unused
x an object of class financials
getFinancials

type  type of statement to view
period period of statement to view
subset ‘xts’ style subset string

Details

A utility to download financial statements for publically traded companies. The data is directly from Google finance. All use of the data is under there Terms of Service, available at http: //www.google.com/accounts/TOS.

Individual statements can be accessed using standard R list extraction tools, or by using viewFinancials. viewFinancials allows for the use of date subsetting as available in the xts package, as well as the specification of the type of statement to view. BS for balance sheet, IS for income statement, and CF for cash flow statement. The period argument is used to identify which statements to view - (A) for annual and (Q) for quarterly.

Value

Six individual matrices organized in a list of class ‘financials’:

IS a list containing (Q)uarterly and (A)nual Income Statements
BS a list containing (Q)uarterly and (A)nual Balance Sheets
CF a list containing (Q)uarterly and (A)nual Cash Flow Statements

Note

As with all free data, you may be getting exactly what you pay for.

Author(s)

Jeffrey A. Ryan

References

Google Finance BETA: http://finance.google.com/finance

Examples

```r
## Not run:
JAVA <- getFinancials('JAVA')
AAPL <- getFin('AAPL')

JAVA$IS$Q  # Quarterly Income Statement
AAPL$CF$A  # Annual Cash Flows

str(AAPL)
## End(Not run)
```
getMetals

Description

Download daily metals prices from oanda.

Usage

getMetals(Metals,
    from = "2007-01-01", to = Sys.Date(),
    base.currency="USD",
    env = .GlobalEnv,
    verbose = FALSE,
    warning = TRUE,
    auto.assign = TRUE, ...)

Arguments

Metals  metals expressed in common name or symbol form
from    start date expressed in ISO CCYY-MM-DD format
to      end date expressed in ISO CCYY-MM-DD format
base.currency which currency should the price be in
env      which environment should they be loaded into
verbose  be verbose
warning  show warnings
auto.assign use auto.assign
...      additional parameters to be passed to getSymbols.oanda method

Details

A convenience wrapper to getSymbols(x,src='oanda').

The most useful aspect of getMetals is the ability to specify the Metals in terms of underlying 3 character symbol or by name (e.g. XAU (gold), XAG (silver), XPD (palladium), or XPT (platinum)).

There are unique aspects of any continuously traded commodity, and it is recommended that the user visit http://www.oanda.com for details on specific pricing issues.

See getSymbols and getSymbols.oanda for more detail.

Value

Data will be assigned automatically to the environment specified (global by default). If auto.assign is set to FALSE, the data from a single metal request will simply be returned from the function call.

If auto.assign is used (the default) a vector of downloaded symbol names will be returned.

See getSymbols and getSymbols.oanda for more detail.
getModelData

Update model's dataset

Description

Update currently specified or built model with most recent data.

Usage

getModelData(x, na.rm = TRUE)

Arguments

x

An object of class quantmod

na.rm

Boolean. Remove NA values. Defaults to TRUE

Details

Primarily used within specify model calls, getModelData is used to retrieve the appropriate underlying variables, and apply model specified transformations automatically. It can be used to also update a current model in memory with the most recent data.

Value

Returns object of class quantmod.OHLC

Author(s)

Jeffrey Ryan
getQuote

See Also
getSymbols, load data, specifyModel, create model structure, buildModel, construct model, modelData, extract model dataset

Examples

```r
## Not run:
my.model <- specifyModel(Next(OpCl(QQQQ)) ~ Lag(Cl(NDX),0:5))
getModelData(my.model)
## End(Not run)
```

getQuote

Download Current Stock Quote

Description

Fetch current stock quote(s) from specified source. At present this only handle sourcing quotes from Yahoo Finance, but it will be extended to additional sources over time.

Usage

```r
getQuote(Symbols, src = "yahoo", what = standardQuote(), ...)
```

```r
standardQuote(src="yahoo")
```

```r
yahooQF(names)
```

Arguments

- **Symbols**: character string of symbols, seperated by semi-colons
- **src**: source of data (only yahoo is implemented)
- **what**: what should be retrieved
- **names**: which data should be retrieved
- **...**: currently unused

Value

A maximum of 200 symbols may be requested per call to Yahoo!, and all requested will be returned in one data.frame object.

getQuote returns a data frame with rows matching the number of Symbols requested, and the columns matching the requested columns.

The what argument allows for specific data to be requested from yahoo, using Yahoo!’s formatting string. A list and interactive selection tool can be seen with yahooQF.

standardQuote currently only applied to Yahoo! data, and returns an object of class quoteFormat, for use within the getQuote function.

Author(s)

Jeffrey A. Ryan
getSymbols.FRED

Download Federal Reserve Economic Data - FRED(R)

Description

R access to over 11,000 data series accessible via the St. Louis Federal Reserve Bank’s FRED system.

Downloads Symbols to specified env from ‘research.stlouisfed.org’. This method is not to be called directly, instead a call to getSymbols(Symbols, src='FRED') will in turn call this method. It is documented for the sole purpose of highlighting the arguments accepted, and to serve as a guide to creating additional getSymbols ‘methods’.

Usage

getSymbols.FRED(Symbols, 
                  env, 
                  return.class = "xts", 
                  ...) 

Arguments

Symbols a character vector specifying the names of each symbol to be loaded 
env where to create objects (.GlobalEnv) 
return.class class of returned object 
... additional parameters 

Details

Meant to be called internally by getSymbols (see also).

One of many methods for loading data for use with quantmod. Essentially a simple wrapper to the underlying FRED data download site.

Naming conventions must follow those as seen on the Federal Reserve Bank of St Louis’s website for FRED. A lookup facility will hopefully be incorporated into quantmod in the near future.

References

Yahoo! Finance finance.yahoo.com gummy-stuff.org www.gummy-stuff.org/Yahoo-data.htm

See Also

getSymbols

Examples

## Not run:
getQuote("AAPL")
getQuote("QQQ;SPY;^VXN", what=yahooQF(c("Bid","Ask")))
standardQuote()
yahooFQ()
## End(Not run)
Value

A call to getSymbols.FRED will load into the specified environment one object for each Symbol specified, with class defined by return.class. Presently this may be ts, lts, zoo, xts, or timeSeries.

Author(s)

Jeffrey A. Ryan

References

St. Louis Fed: Economic Data - FRED http://research.stlouisfed.org/fred2/

See Also

getSymbols, setSymbolLookup

Examples

```r
## Not run:
# All 3 getSymbols calls return the same
# CPI data to the global environment
# The last example is what NOT to do!

## Method #1
getSymbols('CPIAUCNS', src='FRED')

## Method #2
setDefault(getSymbols, src='FRED')
  # OR
  setSymbolLookup(CPIAUCNS='FRED')

getSymbols('CPIAUCNS')

##############################################
## NOT RECOMMENDED!!!
##############################################

## Method #3
getSymbols.FRED('CPIAUCNS', env=globalenv())
## End(Not run)
```

getSymbols.MySQL

Retrieve Data from MySQL Database

Description

Fetch data from MySQL database. As with other methods extending the getSymbols function, this should NOT be called directly. Its documentation is meant to highlight the formal arguments, as well as provide a reference for further user contributed data tools.
getSymbols.MySQL

Usage

getSymbols.MySQL(Symbols,
   env,
   return.class = 'xts',
   db.fields = c("date", "o", "h", "l", "c", "v", "a"),
   field.names = NULL,
   user = NULL,
   password = NULL,
   dbname = NULL,
   ...)

Arguments

Symbols a character vector specifying the names of each symbol to be loaded
env where to create objects. (.GlobalEnv)
return.class desired class of returned object. Can be xts, zoo, data.frame, ts, or its. (zoo)
db.fields character vector indicating names of fields to retrieve
field.names names to assign to returned columns
user username to access database
password password to access database
dbname database name
... currently not used

Details

Meant to be called internally by getSymbols (see also)

One of a few currently defined methods for loading data for use with quantmod. Its use requires the packages DBI and MySQL, along with a running MySQL database with tables corresponding to the Symbol name.

The purpose of this abstraction is to make transparent the ‘source’ of the data, allowing instead the user to concentrate on the data itself.

Value

A call to getSymbols.MySQL will load into the specified environment one object for each Symbol specified, with class defined by return.class.

Note

The default configuration needs a table named for the Symbol specified (e.g. MSFT), with column names date,o,h,l,c,v,a. For table layout changes it is best to use setDefaults(getSymbols.MySQL,...) with the new db.fields values specified.

Author(s)

Jeffrey A. Ryan
getSymbols

Load and Manage Data from Multiple Sources

Description

Functions to load and manage Symbols in specified environment. Used by specifyModel to retrieve symbols specified in first step of modelling procedure. Not a true S3 method, but methods for different data sources follow an S3-like naming convention. Additional methods can be added by simply adhering to the convention.

Current src methods available are: yahoo, google, MySQL, FRED, csv, RData, and Oanda.

Data is loaded silently without user assignment by default.
Usage

getSymbols(Symbols = NULL,
    env = .GlobalEnv,
    reload.Symbols = FALSE,
    verbose = FALSE,
    warnings = TRUE,
    src = "yahoo",
    symbol.lookup = TRUE,
    auto.assign = TRUE,
    ...
)

showSymbols(env=.GlobalEnv)
removeSymbols(Symbols=NULL,env=.GlobalEnv)
saveSymbols(Symbols = NULL,
    file.path=stop("must specify 'file.path'"),
    env = .GlobalEnv)

Arguments

Symbols a character vector specifying the names of each symbol to be loaded
env where to create objects. (.GlobalEnv)
reload.Symbols boolean to reload current symbols in specified environment. (FALSE)
verbose boolean to turn on status of retrieval. (FALSE)
warnings boolean to turn on warnings. (TRUE)
src character string specifying sourcing method. (yahoo)
symbol.lookup retrieve symbol’s sourcing method from external lookup (TRUE)
auto.assign should results be loaded to the environment
file.path character string of file location
... additional parameters

Details

getSymbols is a wrapper to load data from different sources - be them local or remote. Data
is fetched through one of the available getSymbols methods and saved in the env specified -
the .GlobalEnv by default. Data is loaded in much the same way that load behaves. By default,
it is assigned automatically to a variable in the specified environment, without the user explicitly
assigning the returned data to a variable.

The previous sentence’s point warrants repeating - getSymbols is called for its side effects, and
does not return the data object loaded. The data is ‘loaded’ silently by the function into the user’s
environment - or an environment specified. This behavior can be overridden by setting auto.assign
to FALSE, though it is not advised.

By default the variable chosen is an R-legal name derived from the symbol being loaded. It is
possible, using setSymbolLookup to specify an alternate name if the default is not desired, see
that function for details.

The result of a call to getSymbols when auto.assign is set to TRUE (the default) is a new object
or objects in the user’s specified environment - with the loaded symbol(s) names returned upon exit.
getSymbols

If auto.assign is set to FALSE the data will be returned from the call, and will require the user to assign the results himself.

Most, if not all, documentation and functionality in quantmod assumes that auto.assign remains set to TRUE.

Upon completion a list of loaded symbols is stored in the global environment under the name .getSymbols.

Objects loaded by getSymbols with auto.assign=TRUE can be viewed with showSymbols and removed by a call to removeSymbols. Additional data loading “methods” can be created simply by following the S3-like naming convention where getSymbols.NAME is used for your function NAME. See getSymbols source code.

setDefaults(getSymbols) can be used to specify defaults for getSymbols arguments. setDefaults(getSymbols.MySQL) may be used for arguments specific to getSymbols.MySQL, etc.

The “sourcing” of data is managed internally through a complex lookup procedure. If symbol.lookup is TRUE (the default), a check is made if any symbol has had its source specified by setSymbolLookup.

If not set, the process continues by checking to see if src has been specified by the user in the function call. If not, any src defined with setDefaults(getSymbols,src=) is used.

Finally, if none of the other source rules apply the default getSymbols src method is used (‘yahoo’).

Value

A call to getSymbols will load into the specified environment one object for each Symbol specified, with class defined by return.class. Presently this may be ts, its, zoo, xts, or timeSeries.

If auto.assign is set to FALSE an object of type return.class will be returned.

Note

While it is possible to load symbols as classes other than zoo, quantmod requires most, if not all, data to be of class zoo or inherited from zoo - e.g. xts. The additional methods are meant mainly to be of use for those using the functionality outside of the quantmod workflow.

Author(s)

Jeffrey A. Ryan

See Also


Examples

```r
## Not run:
setSymbolLookup(QQQQ='yahoo',SPY='MySQL')
getSymbols(c('QQQQ','SPY'))
# loads QQQQ from yahoo (set with setSymbolLookup)
# loads SPY from MySQL (set with setSymbolLookup)
```
getSymbols('F')
# loads Ford market data from yahoo (the formal default)

setDefaults(getSymbols,verbose=TRUE,src='MySQL')
getSymbols('DIA')
# loads symbol from MySQL database (set with setDefaults)

getSymbols('F',src='yahoo',return.class='ts')
# loads Ford as time series class ts

## End(Not run)

getSymbols.SQLite  Retrieve Data from SQLite Database

Description

Fetch data from SQLite database. As with other methods extending getSymbols this function should NOT be called directly.

Usage

getSymbols.SQLite(Symbols,
                   env,
                   return.class = 'xts',
                   db.fields = c("row_names",
                                 "Open",
                                 "High",
                                 "Low",
                                 "Close",
                                 "Volume",
                                 "Adjusted"),
                   field.names = NULL,
                   dbname = NULL,
                   POSIX = TRUE,
                   ...
)

Arguments

Symbols  a character vector specifying the names of each symbol to be loaded
env      where to create the objects
return.class  desired class of returned object
db.fields  character vector naming fields to retrieve
field.names  names to assign to returned columns
dbname  database name
POSIX  are rownames numeric
...  additional arguments
getSymbols.csv

Details

Meant to be called internally by `getSymbols` (see also)

One of a few currently defined methods for loading data for use with `quantmod`. Its use requires the packages 'DBI' and 'RSQLite', along with a SQLite database.

The purpose of this abstraction is to make transparent the 'source' of the data, allowing instead the user to concentrate on the data itself.

Value

A call to `getSymbols.SQLite` will load into the specified environment one object for each 'Symbol' specified, with class defined by 'return.class'.

Note

This function is experimental at best, and has not been thoroughly tested. Use with caution, and please report any bugs to the maintainer of quantmod.

Author(s)

Jeffrey A. Ryan

References

`SQLite <http://www.sqlite.org>`

David A. James RSQLite: SQLite interface for R

R-SIG-DB. DBI: R Database Interface

See Also

`getSymbols`

Examples

```r
## Not run:
getSymbols("QQQQ", src="SQLite")
## End(Not run)
```

*Description*

Downloads `Symbols` to specified `env` from local comma seperated file. This method is not to be called directly, instead a call to `getSymbols(Symbols, src='csv')` will in turn call this method. It is documented for the sole purpose of highlighting the arguments accepted, and to serve as a guide to creating additional `getSymbols` 'methods'.

---

**getSymbols.csv**  
**Load Data from csv File**

Description

Downloads `Symbols` to specified `env` from local comma seperated file. This method is not to be called directly, instead a call to `getSymbols(Symbols, src='csv')` will in turn call this method. It is documented for the sole purpose of highlighting the arguments accepted, and to serve as a guide to creating additional `getSymbols` 'methods'.
getSymbols.csv

Usage

getSymbols.csv(Symbols,
   env,
   dir="",
   return.class = "xts",
   extension="csv",
   ...)

Arguments

Symbols a character vector specifying the names of each symbol to be loaded
env where to create objects. (.GlobalEnv)
dir directory of csv file
return.class class of returned object
extension extension of csv file
... additional parameters

Details

Meant to be called internally by getSymbols (see also).

One of a few currently defined methods for loading data for use with quantmod. Essentially a simple wrapper to the underlying R read.csv.

Value

A call to getSymbols.csv will load into the specified environment one object for each Symbol specified, with class defined by return.class. Presently this may be ts, its, zoo, xts, or timeSeries.

Note

This has yet to be tested on a windows platform. It should work though file separators may be an issue.

Author(s)

Jeffrey A. Ryan

See Also

getSymbols, read.csv, setSymbolLookup

Examples

```r
## Not run:
# All 3 getSymbols calls return the same
# MSFT to the global environment
# The last example is what NOT to do!

## Method #1
getSymbols('MSFT', src='csv')
```
## Method #2
setDefaults(getSymbols, src='csv')
# OR
setSymbolLookup(MSFT='csv')

getSymbols('MSFT')

#########################################
## NOT RECOMMENDED!!!
#########################################
## Method #3
getSymbols.csv('MSFT', verbose=TRUE, env=globalenv())
## End(Not run)

---

### getSymbols.google

**Download OHLC Data From Google Finance**

**Description**

Downloads Symbols to specified env from ‘finance.google.com’. This method is not to be called directly, instead a call to `getSymbols(Symbols, src='google')` will in turn call this method. It is documented for the sole purpose of highlighting the arguments accepted, and to serve as a guide to creating additional getSymbols ‘methods’.

**Usage**

```r
getSymbols.google(Symbols,
  env,
  return.class = 'xts',
  from = "2007-01-01",
  to = Sys.Date(),
  ...)
```

**Arguments**

- **Symbols**: a character vector specifying the names of each symbol to be loaded
- **env**: where to create objects (GlobalEnv)
- **return.class**: class of returned object
- **from**: Retrieve no earlier than this date
- **to**: Retrieve through this date
- **...**: additional parameters

**Details**

Meant to be called internally by `getSymbols` (see also).

One of a few currently defined methods for loading data for use with `quantmod`. Essentially a simple wrapper to the underlying Google Finance site for historical data.

A word of warning. Google is the home of BETA, and historic data is no exception. There is a BUG in practically all series that include the dates Dec 29, 30, and 31 of 2003. The data will show the wrong date and corresponding prices. This essentially makes it useless, but if they ever apply a fix the data is nice(r) than Yahoo, in so much as it is all split adjusted and there is forty years worth to be had. As long as you skip the holiday week of 2003. : )
getSymbols.google

Value

A call to getSymbols.google will load into the specified environment one object for each Symbol specified, with class defined by return.class. Presently this may be ts, its, zoo, xts, or timeSeries.

Note

As mentioned in the details section, a serious flaw exists within the google database/SQL. A caution is issued when retrieving data via this method if this particular error is encountered, but one can only wonder what else may be wrong. Caveat emptor.

Author(s)

Jeffrey A. Ryan

References

Google Finance: http://finance.google.com

See Also

getSymbols, setSymbolLookup

Examples

## Not run:
# All 3 getSymbols calls return the same
# MSFT to the global environment
# The last example is what NOT to do!

## Method #1
getSymbols('MSFT', src='google')

## Method #2
setDefaults(getSymbols, src='google')

### OR
setSymbolLookup(MSFT='google')

getSymbols('MSFT')

# NOT RECOMMENDED!!!

## Method #3
getSymbols.google('MSFT', verbose=TRUE, env=globalenv())

## End(Not run)
getSymbols.oanda Download Currency and Metals Data from Oanda.com

Description
Access to 191 currency and metal prices, downloadable as more than 36000 currency pairs from Oanda.com.

Downloads Symbols to specified env from www.oanda.com historical currency database. This method is not meant to be called directly, instead a call to getSymbols(x, src="oanda") will in turn call this method. It is documented for the sole purpose of highlighting the arguments accepted, and to serve as a guide to creating additional getSymbols 'methods'.

Usage
getSymbols.oanda(Symbols,
    env,
    return.class = "xts",
    from = "2007-01-01",
    to = Sys.Date(),
    ...
)

Arguments
Symbols a character vector specifying the names of each symbol to be loaded - expressed as a currency pair. (e.g. U.S. Dollar to Euro rate would be expressed as a string "USD/EUR". The naming convention follows from Oanda.com, and a table of possible values is available by calling oanda.currencies

env where to create objects.

return.class class of returned object

from Start of series expressed as "CCYY-MM-DD"

to Start of series expressed as "CCYY-MM-DD"

... additional parameters

Details
Meant to be called internally by getSymbols only.

Oanda data is 7 day daily average price data, that is Monday through Sunday. There is a limit of 2000 days per request, and getSymbols will fail with a warning that the limit has been exceeded.

Value
A call to getSymbols(Symbols, src="oanda") will load into the specified environment one object for each 'Symbol' specified, with class defined by 'return.class'. Presently this may be 'ts', 'its', 'zoo', 'xts', or 'timeSeries'.

Note
Oanda rates are quoted as one unit of base currency to the equivalent amount of foreign currency.
getSymbols.rda

Author(s)
Jeffrey A. Ryan

References
Oanda.com http://www.oanda.com

See Also
Currencies: getSymbols.FRED, getSymbols

Examples

```r
## Not run:
getSymbols("USD/EUR",src="oanda")
getSymbols("USD/EUR",src="oanda",from="2005-01-01")
## End(Not run)
```

getSymbols.rda

Load Data from R Binary File

Description
Downloads Symbols to specified env from local R data file. This method is not to be called directly, instead a call to getSymbols(Symbols, src='rda') will in turn call this method. It is documented for the sole purpose of highlighting the arguments accepted, and to serve as a guide to creating additional getSymbols 'methods'.

Usage

```r
getSymbols.rda(Symbols,
    env,
    dir="",
    return.class = "xts",
    extension="rda",
    col.names=c("Open","High","Low","Close","Volume","Adjusted"),
    ...)
```

Arguments

- **Symbols**: a character vector specifying the names of each symbol to be loaded
- **env**: where to create objects (.GlobalEnv)
- **dir**: directory of rda/RData file
- **return.class**: class of returned object
- **extension**: extension of R data file
- **col.names**: data column names
- **...**: additional parameters
getSymbols.yahoo

Details

Meant to be called internally by getSymbols (see also).

One of a few currently defined methods for loading data for use with quantmod. Essentially a simple wrapper to the underlying R load.

Value

A call to getSymbols.csv will load into the specified environment one object for each Symbol specified, with class defined by return.class. Presently this may be ts, its, zoo, xts, data.frame, or timeSeries.

Author(s)

Jeffrey A. Ryan

See Also

getSymbols, load, setSymbolLookup

Examples

## Not run:
# All 3 getSymbols calls return the same
# MSFT to the global environment
# The last example is what NOT to do!

## Method #1
getSymbols('MSFT', src='rda')
getSymbols('MSFT', src='RData')

## Method #2
setDefault(getSymbols, src='rda')
  # OR
setSymbolLookup(MSFT='rda')
  # OR
setSymbolLookup(MSFT=list(src='rda'))

getSymbols('MSFT')

########################################################################
# NOT RECOMMENDED!!!
########################################################################
## Method #3
getSymbols.rda('MSFT', verbose=TRUE, env=globalenv())
## End(Not run)
getSymbols.yahoo

Description

Downloads Symbols to specified env from ‘finance.yahoo.com’. This method is not to be called directly, instead a call to `getSymbols(Symbols, src='yahoo')` will in turn call this method. It is documented for the sole purpose of highlighting the arguments accepted, and to serve as a guide to creating additional getSymbols ‘methods’.

Usage

```r
getSymbols.yahoo(Symbols,
    env,
    return.class = 'xts',
    from = "2007-01-01",
    to = Sys.Date(),
    ...
)
```

Arguments

- `Symbols`: a character vector specifying the names of each symbol to be loaded
- `env`: where to create objects. (.GlobalEnv)
- `return.class`: class of returned object
- `from`: Retrieve data no earlier than this date. (1990-01-01)
- `to`: Retrieve data through this date (Sys.Date())
- `...`: additional parameters

Details

Meant to be called internally by `getSymbols` (see also).

One of a few currently defined methods for loading data for use with `quantmod`. Essentially a simple wrapper to the underlying Yahoo! finance site’s historical data download.

Value

A call to `getSymbols.yahoo` will load into the specified environment one object for each Symbol specified, with class defined by `return.class`. Presently this may be `ts, its, zoo, xts`, or `timeSeries`.

Author(s)

Jeffrey A. Ryan

References

Yahoo Finance: [http://finance.yahoo.com](http://finance.yahoo.com)

See Also

`getSymbols, setSymbolLookup`
Examples

```r
## Not run:
# All 3 getSymbols calls return the same
# MSFT to the global environment
# The last example is what NOT to do!

## Method #1
getSymbols('MSFT', src='yahoo')

## Method #2
setDefaults(getSymbols, src='yahoo')
  # OR
setSymbolLookup(MSFT='yahoo')

getSymbols('MSFT')

#########################################
## NOT RECOMMENDED!!
#########################################
## Method #3
getSymbols.yahoo('MSFT', env=globalenv())
## End(Not run)
```

has.OHLC

Check For OHLC Data

Description

A set of functions to check for appropriate OHLC and HLC column names within a data object, as well as the availability and position of those columns.

Usage

```r
is.OHLC(x)
has.OHLC(x, which = FALSE)

is.HLC(x)
has.HLC(x, which = FALSE)

has.Op(x, which = FALSE)
has.Hi(x, which = FALSE)
has.Lo(x, which = FALSE)
has.Cl(x, which = FALSE)
has.Vo(x, which = FALSE)
has.Ad(x, which = FALSE)
```

Arguments

- `x`  
  data object

- `which`  
  dispaly position of match
Details

Mostly used internally by quantmod, they can be useful for checking whether an object can be used in OHLC requiring functions like Op, OpCl, etc.

Columns names must contain the full description of data, that is, Open, High, Low, Close, Volume or Adjusted. Abbreviations will return FALSE (or NA when which=TRUE). See quantmod.OHLC for details of quantmod naming conventions.

is.OHLC (and is.HLC, similarly) will only return TRUE is there are columns for Open, High, Low and Close. Additional columns will not affect the value.

Value

A logical value indicating success or failure by default.

If which=TRUE, a numeric value representing the column position will be returned.

is.OHLC and is.HLC return a single value of TRUE or FALSE.

Author(s)

Jeffrey A. Ryan

See Also

quantmod.OHLC,OHLC.Transformations

Examples

```r
## Not run:
getSymbols("YHOO")

is.OHLC(YHOO)
has.OHLC(YHOO)

has.Ad(YHOO)

## End(Not run)
```
is.quantmod

Test If Object of Type quantmod

Description

Test if object is of type quantmod or quantmodResults.

Usage

is.quantmod(x)

is.quantmodResults(x)

Arguments

x object to test

Value

Boolean TRUE or FALSE

Author(s)

Jeffrey A. Ryan

See Also

specifyModel, tradeModel

modelData

Extract Dataset Created by specifyModel

Description

Extract from a quantmod object the dataset created for use in modelling.

specifyModel creates a zoo object for use in subsequent workflow stages (buildModel, tradeModel) that combines all model inputs, from a variety of sources, into one model frame.

modelData returns this object.

Usage

modelData(x, data.window = NULL, exclude.training = FALSE)

Arguments

x a quantmod object

data.window a character vector of subset start and end dates to return

exclude.training

remove training period
Details

When a model is created by \code{specifyModel}, it is attached to the returned object. One of the slots of this S4 class is \code{model.data}.

Value

an object of class \code{zoo} containing all transformations to data specified in \code{specifyModel}.

Author(s)

Jeffrey A. Ryan

See Also

\code{specifyModel,getModelData}

Examples

```r
## Not run:
m <- specifyModel(Next(OpCl(SPY)) ~ Cl(SPY) + OpHi(SPY) + Lag(Cl(SPY)))
modelData(m)
## End(Not run)
```

---

modelSignal \hspace{1cm} \textit{Extract Model Signal Object}

Description

Extract model signal object from \code{quantmodResults} object as an object of class \code{zoo}.

Usage

\code{modelSignal(x)}

Arguments

\code{x} \hspace{1cm} object of class \code{quantmodResults}

Details

For use after a call to \code{tradeModel} to extract the generated signal of a given \code{quantmod} model. Normally this would not need to be called by the end user unless he was manually post processing the trade results.

Value

A \code{zoo} object indexed by signal dates.

Author(s)

Jeffrey A. Ryan
newTA

See Also

tradeModel

newTA  Create A New TA Indicator For chartSeries

Description

Functions to assist in the creation of indicators or content to be drawn on plots produced by chart-Series.

Usage

```r
addTA(ta,
      order = NULL,
      on = NA,
      legend = "auto",
      ...)
```

```r
newTA(FUN,
      tFUN,
      on = NA,
      legend.name,
      fdots = TRUE,
      cdots = TRUE,
      data.at = 1,
      ...)
```

Arguments

ta  data to be plotted
order  which should the columns (if > 1) be plotted
legend  what custom legend text should be added to the chart.
FUN  Main filter function name - as a symbol
tFUN  Pre-filter transformation or extraction function
on  where to draw
legend.name  optional legend heading, automatically derived otherwise
fdots  should any ... be included in the main filter call
cdots  should any ... be included in the resultant function object. fdots=TRUE will override this to TRUE.
data.at  which argument to the main filter function is for data.
...  any additional graphical parameters/default to be included.
Details

Both `addTA` and `newTA` can be used to dynamically add custom content to a displayed chart.

`addTA` takes a series of values, either in a form coercible to `xts` or of the same length as the charted series has rows, and displays the results in either a new TA sub-window, or over/underlaid on the main price chart. If the object can be coerced to `xts`, the time values present must only be within the original series time-range. Internally a merge of dates occurs and will allow for the plotting of discontinuous series.

The `order` argument allows for multiple column data to be plotted in an order that makes the most visual sense.

Specifying a `legend` will override the standard parsing of the `addTA` call to attempt a guess at a suitable title for the sub-chart. Specifying this will cause the standard last value to not be printed.

The `...arg` to `addTA` is used to set graphical parameters interpretable by `lines`.

`newTA` acts as more of a skeleton function, taking functions as arguments, as well as charting parameters, and returns a function that can be called in the same manner as the built-in TA tools, such as `addRSI` and `addMACD`. Essentially a dynamic code generator that allows for highly customizable chart tools with minimal (possibly zero) coding. It is also possible to modify the resultant code to further change behavior.

To create a new TA function with `newTA` certain arguments must be specified.

The `FUN` argument is a function symbol (or coercible to such) that is the primary filter to be used on the core-data of a chartSeries chart. This can be like most of the functions within the TTR package — e.g. RSI or EMA. The resultant object of the function call will be equal to calling the function on the original data passed into the chartSeries function that created the chart. It should be coercible to a matrix object, of one or more columns of output. By default all columns of output will be added to the chart, unless suppressed by passing the appropriately positioned `type='n'` as the `...arg`. Note that this will not suppress the labels added to the chart.

The `tFUN` argument will be called on the main chart's data prior to passing it to `FUN`. This must be a function symbol or a character string of the name function to be called.

The `on` is used to identify which subchart to add the graphic to. By default, `on=NA` will draw the series in a new subchart below the last indicator. Setting this to either a positive or negative value will allow for the series to be super-imposed on, or under, the (sub)chart specified, respectively. A value of 1 refers to the main chart, and at present is the only location supported.

`legend.name` will change the main label for a new plot.

`fdots` and `cdots` enable inclusion or suppression of the `...` within the resulting TA code’s call to `FUN`, or the argument list of the new TA function, respectively. In order to facilitate user-specified graphical parameters it is usually desirable to not impose artificial limits on the end-user with constraints on types of parameters available. By default the new TA function will include the dots argument, and the internal `FUN` call will keep all arguments, including the dots. This may pose issues if the internal function then passes those `...` arguments to a function that can’t handle them.

The final argument is `data.at` which is the position in the `FUN` argument list which expects the data to be passed in at. This default to the sensible first position, though can be changed at the time of creation by setting this argument to the required value.

While the above functions are usually sufficient to construct very pleasing graphical additions to a chart, it may be necessary to modify by-hand the code produced. This can be accomplished by dumping the function to a file, or using `fix` on it during an interactive session.

Another item of note, with respect to `newTA` is the naming of the main legend label. Following `addTA` convention, the first ‘add’ is stripped from the function name, and the rest of the call’s name is used as the label. This can be overridden by specifying `legend.name` in the construction of
the new TA call, or by passing `legend` into the new TA function. Subtle differences exist, with the
former being the preferred solution.

While both functions can be used to build new indicators without any understanding of the internal
chartSeries process, it may be beneficial in more complex cases to have a knowledge of the multi-
step process involved in creating a chart via chartSeries.

to be added...

### Value

`addTA` will invisibly return an S4 object of class `chobTA`. If this function is called interactively,
the `chobTA` object will be evaluated and added to the current chart.

`newTA` will return a function object that can either be assigned or evaluated. Evaluating this func-
tion will follow the logic of any standard addTA-style call, returning invisibly a `chobTA` object, or
adding to the chart.

### Note

Both interfaces are meant to facilitate custom chart additions. `addTA` is for adding any arbitrary
series to a chart, where-as `newTA` works with the underlying series with the main chart object. The
latter also acts as a dynamic TA skeleton generation tool to help develop reusable TA generation
code for use on any chart.

### Author(s)

Jeffrey A. Ryan

### See Also

`chartSeries, TA, chob, chobTA`

### Examples

```r
## Not run:
getSymbols('SBUX')
barChart(SBUX)
addTA(EMA(Cl(SBUX)), on=1, col=6)
addTA(OpCl(SBUX), col=4, type='b', lwd=2)
# create new EMA TA function
newEMA <- newTA(EMA, Cl, on=1, col=7)
newEMA()
newEMA(on=NA, col=5)
## End(Not run)
```

---

### options.expiry

#### Calculate Contract Expirations

**Description**

Return the index of the contract expiration date. The third Friday of the month for options, the last
third Friday of the quarter for futures.
Usage

options.expiry(x)
futures.expiry(x)

Arguments

x

a time-indexed zoo object

Details

Designed to be used within a charting context via addExpiry, the values returned are based on the description above. Exceptions, though rare, are not accounted for.

Value

A numeric vector of values to index on.

Note

There is currently no accounting for holidays that may interfere with the general rule. Additionally all efforts have been focused on US equity and futures markets.

Author(s)

Jeffrey A. Ryan

References

put references to the literature/web site here

See Also

addExpiry

Examples

## Not run:
getSymbols("AAPL")

options.expiry(AAPL)
futures.expiry(AAPL)

AAPL[options.expiry(AAPL)]
## End(Not run)
periodReturn

Calculate Periodic Returns

Description

Given a set of prices, return periodic returns.

Usage

periodReturn(x,
    period='monthly',
    subset=NULL,
    type='arithmetic',
    leading=TRUE,
    ...)

dailyReturn(x, subset=NULL, type='arithmetic',
    leading=TRUE, ...)
weeklyReturn(x, subset=NULL, type='arithmetic',
    leading=TRUE, ...)
monthlyReturn(x, subset=NULL, type='arithmetic',
    leading=TRUE, ...)
quarterlyReturn(x, subset=NULL, type='arithmetic',
    leading=TRUE, ...)
annualReturn(x, subset=NULL, type='arithmetic',
    leading=TRUE, ...)
yearlyReturn(x, subset=NULL, type='arithmetic',
    leading=TRUE, ...)
allReturns(x, subset=NULL, type='arithmetic',
    leading=TRUE)

Arguments

- **x**: object of state prices, or an OHLC type object
- **period**: character string indicating time period. Valid entries are ‘daily’, ‘weekly’, ‘monthly’, ‘quarterly’, ‘yearly’. All are accessible from wrapper functions described below. Defaults to monthly returns (same as monthlyReturn)
- **subset**: an xts/ISO8601 style subset string
- **type**: type of returns: arithmetic (discrete) or log (continuous)
- **leading**: should incomplete leading period returns be returned
- ... passed along to to.period

Details

periodReturn is the underlying function for wrappers:

- allReturns: calculate all available return periods
- dailyReturn: calculate daily returns
- weeklyReturn: calculate weekly returns
quantmod-class

- monthlyReturn: calculate monthly returns
- quarterlyReturn: calculate quarterly returns
- annualReturn: calculate annual returns

Value

Returns object of the class that was originally passed in, with the possible exception of monthly and quarterly return indices being changed to class yearmon and yearqtr where available. This can be overridden with the indexAt argument passed in the to.period function.

By default, if subset is NULL, the full dataset will be used.

Note

Attempts are made to re-convert the resultant series to its original class, if supported by the xts package. At present, objects inheriting from the ‘ts’ class are returned as xts objects. This is to make the results more visually appealing and informative. All xts objects can be converted to class ts with as.ts if that is desirable.

The first and final row of returned object will have the period return to last date, i.e. this week/month/quarter/year return to date even if the start/end is not the start/end of the period. Leading period calculations can be suppressed by setting leading=FALSE.

Author(s)

Jeffrey A. Ryan

See Also

getSymbols to.period

Examples

```r
## Not run:
getSymbols('QQQQ', src='yahoo')
allReturns(QQQQ) # returns all periods

periodReturn(QQQQ, period='yearly', subset='2003::') # returns years 2003 to present
periodReturn(QQQQ, period='yearly', subset='2003') # returns year 2003

rm(QQQQ)
## End(Not run)
```

quantmod-class

Class "quantmod"

Description

Objects of class quantmod help to manage the process of model building within the quantmod package. Created automatically by a call to specifyModel they carry information to be used by a variety of accessor functions and methods.
Objects from the Class

Objects can be created by calls of the form `new("quantmod", ...`). Normally objects are created as a result of a call to `specifyModel`.

Slots

- `model.id`: Object of class "character"
- `model.spec`: Object of class "formula"
- `model.formula`: Object of class "formula"
- `model.target`: Object of class "character"
- `model.inputs`: Object of class "character"
- `build.inputs`: Object of class "character"
- `symbols`: Object of class "character"
- `product`: Object of class "character"
- `price.levels`: Object of class "zoo"
- `training.data`: Object of class "Date"
- `build.date`: Object of class "Date"
- `fitted.model`: Object of class "ANY"
- `model.data`: Object of class "zoo"
- `quantmod.version`: Object of class "numeric"

Methods

No methods defined with class "quantmod" in the signature.

Author(s)

Jeffrey A. Ryan

Examples

`showClass("quantmod")`

---

**quantmod-package**  **Quantitative Financial Modelling Framework**

**Description**

Quantitative Financial Modelling and Trading Framework for R
The quantmod package for R is designed to assist the quantitative trader in the development, testing, and deployment of statistically based trading models.

**What quantmod IS**

A rapid prototyping environment, where quant traders can quickly and cleanly explore and build trading models.

**What quantmod is NOT**

A replacement for anything statistical. It has no ‘new’ modelling routines or analysis tool to speak of. It does now offer charting not currently available elsewhere in R, but most everything else is more of a wrapper to what you already know and love about the language and packages you currently use.

quantmod makes modelling easier by removing the repetitive workflow issues surrounding data management, modelling interfaces, and performance analysis.

**Author(s)**

Jeffrey A. Ryan

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

---

**quantmod.OHLC**

Create Open High Low Close Object

**Description**

Coerce an object with the appropriate columns to class `quantmod.OHLC`, which extends `zoo`.

**Usage**

```r
as.quantmod.OHLC(x, 
   col.names = c("Open", "High", "Low", "Close", "Volume", "Adjusted"), 
   name = NULL, ...)
```
Arguments

setSymbolLookup

Arguments

Arguments

Details

Value

Author(s)

See Also

setSymbolLookup

Manage Symbol Lookup Table

Description

Create and manage Symbol defaults lookup table within R session for use in getSymbols calls.

Usage

Usage

Arguments

Arguments
Details

Use of these functions allows the user to specify a set of default parameters for each Symbol to be loaded.

Different sources (e.g. yahoo, MySQL, csv), can be specified for each Symbol of interest. The sources must be valid getSymbols methods - see getSymbols for details on which methods are available, as well as how to add additional methods.

The argument list to setSymbolLookup is simply the unquoted name of the Symbol matched to the desired default source, or list of Symbol specific parameters.

For example, to signify that the stock data for Sun Microsystems (JAVA) should be downloaded from Yahoo! Finance, one would call setSymbolLookup(JAVA='yahoo') or setSymbolLookup(JAVA=list(src='yahoo'))

It is also possible to specify additional, possibly source specific, lookup details on a per symbol basis. These include an alternate naming convention (useful for sites like Yahoo! where certain non-traded symbols are prepended with a caret, or more correctly a circumflex accent. In that case one would specify setSymbolLookup(DJI=list(name="^DJI",src="yahoo")) as well as passed parameters like dbname and password for database sources. See the specific getSymbols function related to the source in question for more details of each implementation.

All changes are made to the current list, and will persist only until the end of the session. To always use the same defaults it is necessary to call setSymbolLookup with the appropriate parameters from a startup file (e.g. .Rprofile) or to use saveSymbolLookup and loadSymbolLookup to save and restore lookup tables.

To unset a specific Symbol’s defaults, simply assign NULL to the Symbol.

Value

Called for its side effects, the function changes the options value for the specified Symbol through a call to options(getSymbols.sources=...)

Note

Changes are NOT persistent across sessions, as the table is stored in the session options by default. This may change to allow for an easier to manage process, as for now it is designed to minimize the clutter created during a typical session.

Author(s)

Jeffrey A. Ryan

See Also

getSymbols, options.

Examples

setSymbolLookup(QQQQ='yahoo',DIA='MySQL')
getSymbolLookup('QQQQ')
getSymbolLookup(c('QQQQ','DIA'))

## Not run:
## Will download QQQQ from yahoo
## and load DIA from MySQL
getSymbols(c('QQQQ','DIA'))
Description

Used to manage the TA arguments used inside chartSeries calls.

Usage

```
setTA(type = c("chartSeries", "barChart", "candleChart"))
```

Arguments

- `type`: the function to apply defaults TAs to
- `dev`: the device to display TA arguments for

Details

`setTA` and `unsetTA` provide a simple way to reuse the same TA arguments for multiple charts. By default all charting functions will be set to use the current chart’s defaults.

It is important to note that the current device will be used to extract the list of TA arguments to apply. This is done with a call to `listTA` internally, and followed by calls to `setDefaults` of the appropriate functions.

An additional way to set default TA arguments for subsequent charts is via `setDefaults`. See the examples.

Value

Called for its side-effect of setting the default TA arguments to quantmod’s charting functions.

Note

Using `setDefaults` directly will require the vector of default TA calls to be wrapped in a call to `substitute` to prevent unintended evaluations; OR a call to `listTA` to get the present TA arguments. This last approach is what `setTA` wraps.
specifyModel

Specify Model Formula For quantmod Process

Description

Create a single reusable model specification for subsequent buildModel calls. An object of class quantmod is created that can be then be reused with different modelling methods and parameters. No data frame is specified, as data is retrieved from potentially multiple environments, and internal calls to getSymbols.

Usage

specifyModel(formula, na.rm=TRUE)

Arguments

formula an object of class formula (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.

na.rm remove all incomplete rows.

Details

Models are specified through the standard formula mechanism.

As financial models may include a variety of financial and economic indicators, each differing in source, frequency, and/or class, a single mechanism to specify sources is included within a call to specifyModel. See getModelData for details of how this process works.

Currently, objects of class quantmod.OHLC, zoo, ts and its are supported within the model formula.
All symbols are first retrieved from the global environment, without inheritance. If an object is not found in the global environment, it is added to a list of objects to load through the `getSymbols` function. `getSymbols` retrieves each object specified by using information as to its location specified apriori via `setDefaults` or `setSymbolLookup`.

Internally all data is coerced to `zoo`, `data.frame`, or `numeric` classes.

**Value**

Returns an object of class `quantmod`. Use `modelData` to extract full data set as `zoo` object.

**Note**

It is possible to include any supported series in the formula by simply specifying the object’s symbol. See *Details* for a list of currently supported classes.

Use `getSymbols.skeleton` to create additional methods of data sourcing, e.g. from a proprietary data format or currently unimplemented source (Bloomberg, Oracle).

See `getSymbols.MySQL` and `getSymbols.yahoo` for examples of adding additional functionality.

**Author(s)**

Jeffrey Ryan

**References**

`quantmod.com` [http://www.quantmod.com](http://www.quantmod.com)

**See Also**

`getModelData`, `getSymbols`, `buildModel`, `tradeModel`, `formula`, `setSymbolLookup`

**Examples**

```r
## Not run:
# if QQQQ is not in the Global environment, an attempt will be made
# to retrieve it from the source specified with getSymbols.Default

specifyModel(Next(OpCl(QQQQ)) ~ Lag(OpHi(QQQQ),0:3) + Hi(DIA))
```

---

**tradeModel**  

*Simulate Trading of Fitted quantmod Object*

**Description**

Simulated trading of fitted quantmod object. Given a fitted model, `tradeModel` calculates the signal generated over a given historical period, then applies specified `trade.rule` to calculate and return a `tradeLog` object. Additional methods can then be called to evaluate the performance of the model’s strategy.
Usage

```r
tradeModel(x, 
    signal.threshold = c(0, 0), 
    leverage = 1, 
    return.model = TRUE, 
    plot.model = FALSE, 
    trade.dates = NULL, 
    exclude.training = TRUE, 
    ret.type = c("weeks", "months", "quarters", "years"), 
    ...)
```

Arguments

- `x` a quantmod object from `buildModel`
- `signal.threshold` a numeric vector describing simple lower and upper thresholds before trade occurs
- `leverage` amount of leverage to apply - currently a constant
- `return.model` should the full model be returned?
- `plot.model` plot the model?
- `trade.dates` specific trade interval - defaults to full dataset
- `exclude.training` exclude the period trained on?
- `ret.type` a table of period returns
- `...` additional parameters needed by the underlying modelling function, if any

Details

Still highly experimental and changing. The purpose is to apply a newly constructed model from `buildModel` to a new dataset to investigate the model’s trading potential.

At present all parameters are very basic. The near term changes include allowing for a `trade.rule` argument to allow for a dynamic trade rule given a set of signals. Additional the application of variable leverage and costs will become part of the final structure.

Any suggestions as to inclusions or alterations are appreciated and should be directed to the maintainer of the package.

Value

A quantmodResults object

Author(s)

Jeffrey A. Ryan

See Also

`specifyModel` `buildModel`
Examples

```r
## Not run:
m <- specifyModel(Next(OpCl(QQQQ)) ~ Lag(OpHi(QQQQ)))
m.built <- buildModel(m, method='rpart', training.per=c('2007-01-01', '2007-04-01'))
tradeModel(m.built)
tradeModel(m.built, leverage=2)
## End(Not run)
```

Description

Using \textit{xts} style date subsetting, zoom into or out of the current chart.

Usage

```r
zooom(n=1, eps=2)
zoomChart(subset)
```

Arguments

- `n` the number of interactive view changes per call
- `eps` the distance between clicks to be considered a valid subset request
- `subset` a valid subset string

Details

These function allow for viewing of specific areas of a chart produced by \textit{chartSeries} by simply specifying the dates of interest.

\texttt{zooom} is an interactive chart version of \texttt{zoomChart} which utilizes the standard \textit{R} device inter-action tool \texttt{locator} to estimate the subset desired. This estimate is then passed to \texttt{zoomChart} for actual redrawing. At present it is quite experimental in its interface and arguments. Its usage entails a call to \texttt{zooom()} followed by the selection of the leftmost and rightmost points desired in the newly zoomed chart. This selection is accomplished by the user left-clicking each extreme point. Two click are required to determine the level of zooming. Double clicking will reset the chart to the full data range. The arguments and internal working of this function are likely to change dramatically in future releases, though its use will likely remain.

Standard format for the \texttt{subset} argument is the same as the substrsetting for \textit{xts} objects, which is how the data is stored internally for rendering.

Calling \texttt{zoomChart} with no arguments (NULL) resets the chart to the original data.

Examples include '2007' for all of the year 2007, '2007::2008' for years 2007 through 2008, '::2007' for all data from the beginning of the set to the end of 2007, '2007::' all data from the beginning of 2007 through the end of the data. For specifics regarding the level of detail and internal interpretation please see [.\textit{xts}]

Value

This function is called for its side effect - notably changing the perspective of the current chart, and changing its formal subset level. The underlying data attached to the chart is left unchanged.
Author(s)
Jeffrey A. Ryan

See Also
chartSeries

Examples

```r
## Not run:
data(sample_matrix)
chartSeries(sample_matrix)
zoomChart('2007-04::')
zoomChart()

zooom() # interactive example
## End(Not run)
```
Index

* Topic IO
  getQuote, 36

* Topic aplot
  newTA, 56
  TA, 7

* Topic classes
  chob-class, 26
  chobTA-class, 27
  quantmod-class, 61

* Topic datagen
  buildData, 19
  Lag, 2
  Next, 4

* Topic datasets
  getModelData, 35
  getSymbols.oanda, 48

* Topic data
  getQuote, 36
  getSymbols, 40
  getSymbols.csv, 44
  getSymbols.FRED, 37
  getSymbols.google, 46
  getSymbols.MySQL, 38
  getSymbols.rda, 49
  getSymbols.yahoo, 50
  modelData, 54
  quantmod.OHLC, 63

* Topic dplot
  newTA, 56

* Topic hplot
  newTA, 56

* Topic misc
  Lag, 2
  Next, 4

* Topic models
  buildModel, 20
  fittedModel, 28
  specifyModel, 67
  tradeModel, 68

* Topic package
  quantmod-package, 62

* Topic ts
  Lag, 2

* Topic utilities
  addADX, 9
  addBBands, 10
  addEXpiry, 11
  addMA, 11
  addMACD, 13
  addROC, 14
  addRSI, 15
  addSAR, 16
  addSMA, 17
  addVo, 18
  addWPR, 18
  chartSeries, 21
  chartTheme, 24
  Delt, 1
  getDividends, 30
  getFinancials, 32
  getFX, 31
  getMetals, 34
  getSymbols.SQLite, 43
  has.OHLC, 52
  internal-quantmod, 53
  is.quantmod, 54
  modelData, 54
  modelSignal, 55
  OHLC.Transformations, 5
  options.expiry, 58
  periodReturn, 60
  setSymbolLookup, 64
  setTA, 66
  zoomChart, 70

Ad(OHLC.Transformations), 5
addADX, 9
addATR(TA), 7
addBBands, 10
addCCI(TA), 7
addCMF(TA), 7
addCMO(TA), 7
addDEMA(addMA), 11
addDPO(TA), 7
addEMA(addMA), 11
addEnvelope(TA), 7
addEVNMA(addMA), 11
addExpiry, 11, 59
addLines (TA), 7
addMA, 11
addMADCD, 13
addMomentum (TA), 7
addPoints (TA), 7
addROC, 14
addRSI, 15
addSAR, 16
addShading (internal-quantmod), 53
addSMA (addMA), 11
addSMAI, 17
addTA, 9–19, 24, 27, 67
addTRIX (TA), 7
addVo, 18
addWMA (addMA), 11
addWPB, 18
addZLEMA (addMA), 11
allReturns (periodReturn), 60
annualReturn (periodReturn), 60
anova.quantmod (fittedModel), 28
as.quantmod.OHLC (quantmod.OHLC), 63

barChart (chartSeries), 21
buildData, 19
buildModel, 20, 29, 36, 68, 69

candleChart (chartSeries), 21
chartSeries, 21, 25, 27, 58, 67, 71
chartShading (internal-quantmod), 53

chartTheme, 24, 24
chob, 27, 58
chob-class, 26
chobTA, 27, 58
chobTA-class, 27
Cl (OHLC.Transformations), 5
ClCl (OHLC.Transformations), 5
ccoef.quantmod (fittedModel), 28
coefficients.quantmod (fittedModel), 28
dailyReturn (periodReturn), 60
Delt, 1
dropTA (TA), 7

fitted.quantmod (fittedModel), 28
fitted.values.quantmod (fittedModel), 28
fittedModel, 28
fittedModel <- (fittedModel), 28
formula, 68
formula.quantmod (fittedModel), 28
futures.expiry (options.expiry), 58

gDividends, 30
gFin (getFinancials), 32
gFinancials, 32
gFX, 31, 42
gMetals, 34, 42
gModelData, 35, 42, 55, 68
gQuote, 36
gSymbolLookup
gSymbols, 20, 24, 31, 32, 35–38, 40, 40, 44, 45, 47, 49–51, 61, 64, 65, 68
gSymbols.csv, 42, 44
gSymbols.FRED, 37, 42, 49
gSymbols.google, 42, 46
gSymbols.MySQL, 38
gSymbols.mysql
gSymbols.oanda, 32, 35, 42, 48
gSymbols.rda, 49
gSymbols.RData, 42
gSymbols.RData
gSymbols.rda, 49
gSymbols.SQLite, 43
gSymbols.yahoo, 42, 50

has.Ad (has.OHLC), 52
has.Cl (has.OHLC), 52
has.Hi (has.OHLC), 52
has.HLC (has.OHLC), 52
has.Lo (has.OHLC), 52
has.OHLC, 52
has.Op (has.OHLC), 52
has.Vo (has.OHLC), 52
hi (OHLC.Transformations), 5
Hi (OHLC.Transformations), 5
HiCl (OHLC.Transformations), 5
HLC (OHLC.Transformations), 5

internal-quantmod, 53
is.HLC (has.OHLC), 52
is.OHLC (has.OHLC), 52
is.quantmod, 54
is.quantmodResults (is.quantmod), 54

Lag, 2, 5
lag, 3
Lag.quantmod.OHLC (Lag), 2
Lag.zoo (Lag), 2
INDEX

lineChart (chartSeries), 21
listTA (setTA), 66
Lo (OHLC.Transformations), 5
load, 50
loadSymbolLookup
 (setSymbolLookup), 64
LoCl (OHLC.Transformations), 5
logLik.quantmod (fittedModel), 28
LoHi (OHLC.Transformations), 5
matchChart (chartSeries), 21
modelData, 20, 36, 54
modelSignal, 55
monthlyReturn (periodReturn), 60
moveTA (TA), 7
newTA, 56
Next, 4
Next.quantmod.OHLC (Next), 4
Next.zoo (Next), 4

oanda.currencies
 (getSymbols.oanda), 48
OHLC (OHLC.Transformations), 5
OHLC.Transformations, 5, 53, 64
Op (OHLC.Transformations), 5
OpCl, 2
OpCl (OHLC.Transformations), 5
OpHi (OHLC.Transformations), 5
OpLo (OHLC.Transformations), 5
OpOp, 2
OpOp (OHLC.Transformations), 5
options, 65
options.expiry, 58
periodReturn, 60
plot.quantmod (fittedModel), 28
quantmod, 29
quantmod (quantmod-package), 62
quantmod-class, 61
quantmod-package, 62
quantmod-show (quantmod-class), 61
quantmod.OHLC, 53, 63
quarterlyReturn (periodReturn), 60

read.csv, 45
reChart (chartSeries), 21
removeSymbols (getSymbols), 40
resid.quantmod (fittedModel), 28
residuals.quantmod (fittedModel), 28
saveSymbolLookup
 (setSymbolLookup), 64
saveSymbols (getSymbols), 40
seriesHi (OHLC.Transformations), 5
seriesLo (OHLC.Transformations), 5
setSymbolLookup, 38, 40, 42, 45, 47, 50,
51, 64, 68
setTA, 24, 66
show, chobTA-method
 (chobTA-class), 27
showSymbols (getSymbols), 40
specifyModel, 5, 7, 20, 21, 36, 40, 42, 54,
55, 67, 69
standardQuote (getQuote), 36
swapTA (TA), 7
TA, 7, 58
to.period, 61
tradeModel, 21, 54, 56, 68, 68
unsetSymbolLookup
 (setSymbolLookup), 64
unsetTA (setTA), 66
vcov.quantmod (fittedModel), 28
viewFin (getFinancials), 32
viewFinancials (getFinancials), 32
Vo (OHLC.Transformations), 5
weeklyReturn (periodReturn), 60
yahooQF (getQuote), 36
yearlyReturn (periodReturn), 60
zoom (zoomChart), 70
zoomChart, 70
zoom (zoomChart), 70