Package ‘FinancialInstrument’

February 19, 2015

Copyright (c) 2004 - 2014
Maintainer Garrett See <gsee000@gmail.com>
License GPL
Title Financial Instrument Model Infrastructure for R
Type Package
LazyLoad yes
Author Peter Carl, Brian G. Peterson, Garrett See
Description Infrastructure for defining meta-data and relationships for financial instruments.
Version 1.2.0
URL https://r-forge.r-project.org/projects/blotter/
Date 2014-12-15
Depends R (>= 2.12.0), quantmod (>= 0.4-3), zoo (>= 1.7-5), xts
Imports TTR
Suggests foreach, XML (>= 3.96.1.1), testthat, its, timeSeries
NeedsCompilation no
Repository CRAN
Date/Publication 2014-12-16 00:49:25

R topics documented:

FinancialInstrument-package .............................................. 3
.get_rate ................................................................. 7
.to_daily ................................................................. 8
add.defined.by .......................................................... 9
add.identifier ............................................................ 10
buildHierarchy ........................................................... 11
buildRatio ................................................................. 12
buildSpread ............................................................... 13
build_series_symbols ..................................................... 14
R topics documented:

build_spread_symbols .............................................. 15
C2M ................................................................. 16
CompareInstrumentFiles ............................................. 17
currencies ............................................................ 18
exchange_rate ......................................................... 18
expires ................................................................. 19
find.instrument ......................................................... 21
FindCommonInstrumentAttributes .................................. 22
fn_SpreadBuilder ....................................................... 23
formatSpreadPrice ..................................................... 25
format_id .............................................................. 26
future_series .......................................................... 27
getInstrument ........................................................ 28
getSymbols.FI .......................................................... 29
instrument ............................................................. 31
instrument.auto ........................................................ 33
instrument.table ........................................................ 34
instrument_attr ......................................................... 36
is.currency ............................................................. 37
is.currency.name ....................................................... 37
is.instrument .......................................................... 38
is.instrument.name .................................................... 38
load.instruments ....................................................... 39
ls_by_currency ........................................................ 40
ls_by_exp expiry ......................................................... 42
ls_expiries ............................................................. 43
ls_instruments ......................................................... 44
ls_instruments_by ..................................................... 47
ls_strikes ............................................................... 48
ls_underlyings ........................................................ 49
make_spread_id ........................................................ 50
month_cycle2numeric .................................................. 51
next.future_id ........................................................ 52
Notionalize ............................................................. 53
option_series.yahoo ................................................... 54
parse_id ................................................................. 55
parse_suffix ............................................................ 56
redenominate .......................................................... 57
root_contracts ........................................................ 59
saveInstruments ....................................................... 59
saveSymbols.days ...................................................... 60
setSymbolLookup.FI .................................................... 62
sort_ids ............................................................... 63
synthetic ............................................................... 64
to_secBATV ............................................................ 66
update_instruments.instrument ...................................... 67
update_instruments.iShares ......................................... 69
update_instruments.masterDATA .................................... 70
FinancialInstrument-package

Construct, manage and store contract specifications for trading

Description

Transaction-oriented infrastructure for defining tradable instruments based on their contract specifications. Construct and manage the definition of any asset class, including derivatives, exotics and currencies. Potentially useful for portfolio accounting, backtesting, pre-trade pricing and other financial research. Still in active development.

Details

The FinancialInstrument package provides a construct for defining and storing meta-data for tradable contracts (referred to as instruments, e.g., stocks, futures, options, etc.). It can be used to create any asset class and derivatives, across multiple currencies.

FinancialInstrument was originally part of a companion package, blotter, that provides portfolio accounting functionality. Blotter accumulates transactions into positions, then into portfolios and an account. FinancialInstrument is used to contain the meta-data about an instrument, which blotter uses to calculate the notional value of positions and the resulting P&L. FinancialInstrument, however, has plenty of utility beyond portfolio accounting, and was carved out so that others might take advantage of its functionality.

As used here, ‘instruments’ are S3 objects of type ‘instrument’ or a subclass thereof that define contract specifications for price series for a tradable contract, such as corn futures or IBM common stock. When defined as instruments, these objects are extended to include descriptive information and contract specifications that help identify and value the contract.

A simple example of an instrument is a common stock. An instrument can be defined in brief terms with an identifier (e.g., "IBM"). Beyond the primary identifier, additional identifiers may be added as well and will work as ‘aliases’. Any identifier will do – Bloomberg, Reuters-RIC, CUSIP, etc. – as long as it’s unique to the workspace. In addition, a stock price will be denominated in a currency (e.g., "USD") and will have a specific tick size which is the minimum amount that the price can be quoted and transacted in (e.g., $0.01). We also define a ’multiplier’ that is used when calculating the notional value of a position or transaction using a quantity and price (sometimes called a contract multiplier). For a stock it’s usually ‘1’.

More care is needed when dealing with complex instruments, like futures. First, we have to define a future as a root contract. This root is not tradable unto itself, but is used to generate a series of futures which are tradable and expire through time. The root contract will provide an identifier (e.g., ‘C’ for the CME’s corn contract), a denomination currency, a multiplier (one futures contract will cover multiple items) and a minimum tick size. From that definition, a series of expiring contracts can be generated (“C_H08”, “C_Z08”, etc.) by specifying a suffix to be associated with the series, usually something like ‘Z9’ or ‘Mar10’ denoting expiration and year. As you might expect, options
are treated similarly. The package also includes constructors for certain synthetic instruments, such as spreads.

FinancialInstrument doesn’t try to exhaust the possibilities of attributes, so it instead allows for flexibility. If you wanted to add an attribute to tag the exchange the instrument is listed on, just add it when defining the instrument (e.g., future('CL', multiplier=1000, currency="USD", tick_size=.01, exchange="CM"). Or, as you can see, we’ve found it useful to add a field with more slightly more detail, such as description="IBM Common Stock". You can also add attribute after the instrument has been created using instrument_attr as shown in the examples section below.

Defining instruments can be tedious, so we’ve also included a CSV loader, load_instruments, in the package, as well as some functions that will update instruments with data downloaded from the internet. See, e.g., update_instruments.yahoo, update_instruments.TTR, update_instruments.morningstar, update_instruments.ishares. You can also update an instrument using the details of another one with update_instruments.instrument which can be useful for creating a new future_series from an expiring one.

Once you’ve defined all these instruments (we keep hundreds or thousands of them in our environments), you can save the instrument environment using saveInstruments. When you start a fresh R session, you can load your instrument definitions using loadInstruments. We maintain an instrument.RData file that contains definitions for all instruments for which we have market data on disk.

You may want to use setSymbolLookup.FI to define where and how your market data are stored so that getSymbols will work for you.

FinancialInstrument’s functions build and manipulate objects that are stored in an environment named ".instrument" at the top level of the package (i.e. "FinancialInstrument::.instrument") rather than the global environment, .GlobalEnv. Objects may be listed using ls_instruments() (or many other ls_* functions).

We store instruments in their own environment for two reasons. First, it keeps the user’s workspace less cluttered and lowers the probability of clobbering something. Second, it allows the user to save and re-use the .instrument environment in other workspaces. Objects created with FinancialInstrument may be directly manipulated as any other object, but in our use so far we’ve found that it’s relatively rare to do so. Use the getInstrument function to query the contract specs of a particular instrument from the environment.

Author(s)

Peter Carl, Brian G. Peterson, Garrett See,
Maintainer: G See <gsee00@gmail.com>

See Also

xts, quantmod, blotter, PerformanceAnalytics, qmao, and twsInstrument

Examples

## Not run:
# Construct instruments for several different asset classes
# Define a currency and some stocks
require("FinancialInstrument")
currency(c("USD", "EUR")) # define some currencies
stock(c("SPY", "LQD", "IBM", "GS"), currency="USD") # define some stocks
exchange_rate("EURUSD") # define an exchange rate

ls_stocks() # get the names of all the stocks
ls_instruments() # all instruments

getInstrument("IBM")
update_instruments.yahoo(ls_stocks())
update_instruments.TTR(ls_stocks()) # doesn't update ETFs
update_instruments.masterDATA(ls_stocks()) # only updates ETFs
getInstrument("SPY")

## Compare instruments with all.equal.instrument method
all.equal(getInstrument("USD"), getInstrument("USD"))
all.equal(getInstrument("USD"), getInstrument("EUR"))
all.equal(getInstrument("SPY"), getInstrument("LQD"))

## Search for the tickers of instruments that contain words
find.instrument("computer") # IBM
find.instrument("bond") # LQD

## Find only the ETFs; update_instruments.masterDATA added a "Fund.Type" field
## to the ETFs, but not to the stocks
ls_instruments_by("Fund.Type") # all instruments that have a "Fund.Type" field

# build data.frames with instrument attributes
buildHierarchy(ls_stocks(), "Name", "type", "avg.volume")

## before defining a derivative, must define the root (can define the underlying
## in the same step)
future("ES", "USD", multiplier=50, tick_size=0.25,
  underlying_id=synthetic("SPX", "USD", src=list(src=yahoo, name='^GSPC')))

# above, in addition to defining the future root "ES", we defined an instrument
# named "SPX". Using the "src" argument causes setSymbolLookup to be called.
# Using the "src" arg as above is the same as
# setSymbolLookup(SPX=list(src=yahoo, name='^GSPC'))
getSymbols("SPX") # this now works even though the symbol used by
# getSymbols.yahoo is "^GSPC", not "SPX"

## Back to the futures; we can define a future_series
future_series("ES_U2", identifiers=list(ESU2"))
# identifiers are not necessary, but they allow for the instrument to be found
# by more than one name
getInstrument("ES_U2") # this will find the instrument even though the primary_id
# is "ES_U2"
# can also add identifiers later
add.identifier("ES_U2", inhouse="ES_U12")

# can add an arbitrary field with instrument_attr
instrument_attr("ES_U2", "description", "S&P 500 e-mini")
getInstrument("ES_U2")
option_series.yahoo("GS") # define a bunch of options on "GS"
# option root was automatically created
getInstrument(".GS")
# could also find ".GS" by looking for "GS", but specifying type
getInstrument("GS", type='option')

# if you do not know what type of instrument you need to define, try
instrument.auto("ESM3")
getInstrument("ESM3")
instrument.auto("USDJPY")
getInstrument("USDJPY")

instrument.auto("QQQ") # doesn't work as well on ambiguous tickers
getInstrument("QQQ")

# Some functions that make it easier to work with futures
M2C() # Month To Code
M2C()[5]
M2C("may")
C2M() # Code To Month
C2M("J")
C2M()[7]
MC2N("G") # Month Code to Numeric
MC2N("H,K,M")

parse_id("ES_U3")
parse_id("EURUSD")

next_future_id("ES_U2")
next_future_id("ZC_H2", "H,K,N,U,Z")
prev_future_id("CL_H2", 1:12)

sort_ids(ls_instruments()) # sort by expiration date, then alphabetically for
# things that don't expire.

format_id("ES_U2", "CYY")
format_id("ES_U2", "CYY", sep="")
format_id("ES_U2", "MMMMY")

## Saving the instrument environment to disk
tmpdir <- tempdir()
saveInstruments("MyInstruments.RData", dir=tmpdir)
rm_instruments(keep.currencies=FALSE)
lS_instruments() # NULL
loadInstruments("MyInstruments.RData", dir=tmpdir)
lS_instruments()
unlink(tmpdir, recursive=TRUE)

#build a spread:
fn_SpreadBuilder(getSymbols(c("IBM", "SPY"), src='yahoo'))
head(IBM.SPY)
getInstrument("IBM.SPY")
# alternatively, define a spread, then build it
spread(members=c("IBM", "GS", "SPY"), memberratio=c(1, -2, 1))
buildSpread("IBM.GS.SPY") #Since we hadn't yet downloaded "GS", buildSpread
#downloaded it temporarily
chartSeries(IBM.GS.SPY)

## fn_SpreadBuilder will return as many columns as it can
## (Bid, Ask, Mid, or Op, Cl, Ad), but only works on 2 instrument spreads
## buildSpread works with any number of legs, but returns a single price column

getFX("EUR/USD", from=Sys.Date()-499) # download exchange rate from Oanda

IBM.EUR <- redenominate("IBM", "EUR") #price IBM in EUR instead of dollars
chartSeries(IBM, subset=’last 500 days’, TA=NULL)
addTA(Ad(IBM.EUR), on=1, col=’red’)

## End(Not run)

---

### .get_rate

**get an exchange rate series**

**Description**
Try to find exchange rate data in an environment, inverting if necessary.

**Usage**

```
.get_rate(ccy1, ccy2, env = .GlobalEnv)
```

**Arguments**

- **ccy1**: chr name of 1st currency
- **ccy2**: chr name of 2nd currency
- **env**: environment in which to look for data.

**Value**

xts object with as many columns as practicable.

**Author(s)**

Garrett See

**See Also**

- buildRatio
- redenominate
Examples

```r
## Not run:
EURUSD <- getSymbols("EURUSD=x", src='yahoo', auto.assign=FALSE)
USDEUR <- .get_rate("USD","EUR")
head(USDEUR)
head(EURUSD)

## End(Not run)
```

---

### .to_daily

*Extract a single row from each day in an xts object*

**Description**

Extract a single row from each day in an xts object

**Usage**

```
.to_daily(x, EOD_time = "15:00:00")
```

**Arguments**

- `x` xts object of sub-daily data.
- `EOD_time` time of day to use.

**Value**

xts object with daily scale.

**Author(s)**

Garrett See

**See Also**

quantmod::to.daily, quantmod::to.period
Description

Concatenate a string or strings (passed through dots) to the defined.by field of an instrument (separated by semi-colons). Any duplicates will be removed. See Details.

Usage

```r
add.defined.by(primary_ids, ...)
```

Arguments

- `primary_ids` character vector of primary_ids of instruments
- `...` strings, or character vector, or semi-colon delimited string.

Details

If there is already a value for the defined.by attribute of the primary_id instrument, that string will be split on semi-colons and converted to a character vector. That will be combined with any new strings (in ...). The unique value of this new vector will then be converted into a semi-colon delimited string that will be assigned to the defined.by attribute of the primary_ids’ instruments.

Many functions that create or update instrument definitions will also add or update the value of the defined.by attribute of that instrument. If an instrument has been updated by more than one function, it’s defined.by attribute will likely be a semi-colon delimited string (e.g. “TTR:yahoo”).

Value

called for side-effect

Author(s)

Garrett See

See Also

`add.identifier`, `instrument_attr`

Examples

```r
## Not run:
update_instruments.TTR("GS")
getInstrument("GS")$defined.by # TTR
add.defined.by("GS", "gsee", "demo")
add.defined.by("GS", "gsee;demo") # same

## End(Not run)
```
Description

Add an identifier to an instrument unless the instrument already has that identifier.

Usage

```
add.identifier(primary_id, ...)
```

Arguments

- **primary_id**: primary_id of an instrument
- **...**: identifiers passed as regular named arguments.

Value

called for side-effect

Author(s)

Garrett See

See Also

- **instrument_attr**

Examples

```
## Not run:
stock("XXX", currency("USD"))
add.identifier("XXX", yahoo="XXX")
getInstrument("XXX")
add.identifier("XXX", "X3")
all.equal(getInstrument("X3"), getInstrument("XXX")) # TRUE

## End(Not run)
```
**buildHierarchy**

*Construct a hierarchy of instruments useful for aggregation*

**Description**

Construct a hierarchy of instruments useful for aggregation

**Usage**

`buildHierarchy(primary_ids, ...)`

**Arguments**

- `primary_ids` A character vector of instrument primary_ids to be included in the hierarchy list
- `...` character names of instrument attributes in top-down order.

**Value**

Constructs a data.frame that contains the list of assets in the first column and the category or factor for grouping at each level in the following columns

**Author(s)**

Peter Carl, Alexis Petit, Garrett See

**See Also**

`instrument.table`

**Examples**

```r
## Not run:
# rm_instruments(keep.currencies=FALSE)
## Define some stocks
update_instruments.TTR(c("XOM", "IBM", "CVX", "WMT", "GE"), exchange="NYSE")

buildHierarchy(ls_instruments(), "type")
buildHierarchy(ls_stocks(), c("Name", "Sector"))
buildHierarchy(ls_stocks(), "Industry", "MarketCap")
## End(Not run)
```
Description

Calculates time series of ratio of 2 instruments using available data. Returned object will be ratios calculated using Bids, Asks, and Mids, or Opens, Closes, and Adjusteds.

Usage

buildRatio(x, env = .GlobalEnv, silent = FALSE)

Arguments

x    vector of instrument names. e.g. c("SPY","DIA")
env  environment where xts data is stored
silent silence warnings?

Details

x should be a vector of 2 instrument names. An attempt will be made to get the data for both instruments. If there are no xts data stored under either of the names, it will try to return prebuilt data with a call to .get_rate.

If the data are not of the same frequency, or are not of the same type (OHLC, BBO, etc.) An attempt will be made to make them compatible. Preference is given to the first leg.

If the data in x[1] is daily or slower and the data in x[2] is intraday (e.g. if you give it daily OHLC and intraday Bid Ask Mid, it will use all of the OHLC columns of x[1] and only the the End of Day Mid price of the BAM object.

If the data in x[1] is intraday, and the data in x[2] is daily or slower, for each day, the previous closing value of x[2] will be filled forward with na.locf

Value

An xts object with columns of Bid, Ask, Mid OR Open, Close, Adjusted OR Price

Author(s)

Garrett See

See Also

reddenominate buildSpread fn_SpreadBuilder
Examples

```r
## Not run:
syms <- c("SPY","DIA")
getSymbols(syms)
rat <- buildRatio(syms)
summary(rat)

## End(Not run)
```

### buildSpread

`buildSpread` is a function to construct a price/level series for pre-defined multi-leg spread instrument.

#### Description

Build price series for spreads, butterflies, or other synthetic instruments, using metadata of a previously defined synthetic instrument.

#### Usage

```r
buildSpread(spread_id, Dates = NULL, onelot = TRUE, prefer = NULL,
            auto.assign = TRUE, env = .GlobalEnv)
```

```r
buildBasket(spread_id, Dates = NULL, onelot = TRUE, prefer = NULL,
            auto.assign = TRUE, env = .GlobalEnv)
```

#### Arguments

- **spread_id**: The name of the instrument that contains members and memberratio.
- **Dates**: Date range on which to subset. Also, if a member’s data is not available via `getSymbols` will be called, and the values of the from and to arguments will be determined using `.parseISO8601` on Dates.
- **onelot**: Should the series be divided by the first leg's ratio?
- **prefer**: Price column to use to build structure.
- **auto.assign**: Assign the spread? If FALSE, the xts object will be returned.
- **env**: Environment holding data for members as well as where spread data will be assigned.

#### Details

The spread and all legs must be defined instruments.

This function can build multileg spreads such as calendars, butterflies, condors, etc. However, the returned series will be univariate. It does not return multiple columns (e.g. ‘Bid’, ‘Ask’, ‘Mid’) like `fn_SpreadBuilder` does.

`buildBasket` is an alias

TODO: allow for multiplier (divisor) that is a vector.
Value

If `auto.assign` is `FALSE`, a univariate `xts` object. Otherwise, the `xts` object will be assigned to `spread_id` and the `spread_id` will be returned.

Note

This could also be used to build a basket or a strip by using only positive values in `memberratio`.

Author(s)

Brian Peterson, Garrett See

See Also

- `fn_spreadBuilder` spread for instructions on defining the spread

Examples

```r
## Not run:
currency("USD")
stock("SPY","USD",1)
stock("DIA","USD",1)
getSymbols(c("SPY","DIA"))

spread("SPYDIA","USD",c("SPY","DIA"),c(1,-1)) # define it.
buildSpread('SPYDIA') # build it.
head(SPYDIA)

## End(Not run)
```

---

**build_series_symbols**  
*construct a series of symbols based on root symbol and suffix letters*

### Description

The columns needed by this version of the function are `primary_id` and `month_cycle`. `primary_id` should match the `primary_id` of the instrument describing the root contract. `month_cycle` should contain a comma-delimited string describing the month sequence to use, e.g. "F, G, H, J, K, M, N, Q, U, V, X, Z" for all months using the standard futures letters, or "H, M, U, Z" for quarters, or "Mar, Jun, Sep, Dec" for quarters as three-letter month abbreviations, etc. The correct values will vary based on your data source.

### Usage

```r
build_series_symbols(roots, yearlist = c(0, 1))
```
**build_spread_symbols**

**Arguments**

- **roots**: data.frame containing at least columns `primary_id` and `month_cycle`, see Details
- **yearlist**: vector of year suffixes to be applied, see Details

**Details**

TODO add more flexibility in input formats for `roots` #'

**Author(s)**

Brian G. Peterson

**See Also**

[load.instruments](#)

---

**build_spread_symbols**  
*build symbols for exchange guaranteed (calendar) spreads*

**Description**

The columns needed by this version of the function are `primary_id`, `month_cycle`, and `code contracts_ahead`.

**Usage**

```r
build_spread_symbols(data = NULL, file = NULL, outputfile = NULL, 
                      start_date = Sys.Date())
```

**Arguments**

- **data**: data.frame containing at least columns `primary_id`, `month_cycle`, and `code contracts_ahead`, see Details
- **file**: if not NULL, will read input data from the file named by this argument, in the same format as `data`, above
- **outputfile**: if not NULL, will write output to this file as a CSV
- **start_date**: date to start building from, of type `Date` or an ISO-8601 date string, defaults to `Sys.Date`
Details

primary_id should match the primary_id of the instrument describing the root contract.

month_cycle should contain a comma delimited string describing the month sequence to use, e.g. "F, G, H, J, K, M, N, Q, U, V, X, Z" for all months using the standard futures letters, or "H, M, U, Z" for quarters, or "Mar, Jun, Sep, Dec" for quarters as three-letter month abbreviations, etc. The correct values will vary based on your data source.

contracts_ahead should contain a comma-delimited string describing the cycle on which the guaranteed calendar spreads are to be constructed, e.g. '1' for one-month spreads, '1,3' for one and three month spreads, '1,6,12' for 1, 6, and 12 month spreads, etc. For quarterly symbols, the correct contracts_ahead may be something like '1,2,3' for quarterly, bi-annual, and annual spreads.

active_months is a numeric field indicating how many months including the month of the start_date the contract is available to trade. This number will be used as the upper limit for symbol generation.

If type is also specified, it should be a specific instrument type, e.g. 'future_series', 'option_series', 'guaranteed_spread' or 'calendar_spread'.

One of data or file must be populated for input data.

Author(s)

Ilya Kipnis <Ilya.Kipnis<at>gmail.com>

See Also

load.instruments build_series_symbols

---

C2M Month-to-Code and Code-to-Month

Description

Convert month code (used for futures contracts) to abbreviated month name, or convert abbreviated month name to month code

Usage

C2M(code)

M2C(month)

Arguments

code Month code: F, G, H, J, K, M, N, Q, U, V, X, or Z
month Abbreviated month: jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, or dec

Value

corresponding code or month.
CompareInstrumentFiles

Author(s)
Garrett See

See Also
MC2N

Examples
C2M()
C2M("M")
C2M()[6]
M2C()
M2C("Sep")
M2C()[9]

Description
Compare the .instrument environments of two files

Usage
CompareInstrumentFiles(file1, file2, ...)

Arguments
file1 A file containing an instrument environment
file2 Another file containing an instrument environment. If not provided, file1 will be compared against the currently loaded instrument environment.
... Arguments to pass to all.equal.instrument

Details
This will load two instrument files (created by saveInstruments) and find the differences between them. In addition to returning a list of difference that are found, it will produce messages indicating the number of instruments that were added, the number of instruments that were removed, and the number of instruments that are different.

Value
A list that contains the names of all instruments that were added, the names of all instruments that were removed, and the changes to all instruments that were updated (per all.equal.instrument).
Exchange Rate

Author(s)
Garrett See

See Also
save_instruments, all.equal.instrument

Examples

```r
## Not run:
# backup current .instrument environment
bak <- as.list(FinancialInstrument:::instrument, all.names=TRUE)
old.wd <- getwd()
tmpdir <- tempdir()
setwd(tmpdir)
rm_instruments(keep=FALSE)
# create some instruments and save
stock(c("SPY", "DIA", "GLD"), currency("USD"))
saveInstruments("MyInstruments1")
# make some changes
rm_stocks("GLD")
stock("QQQ", "USD")
instrument_attr("SPY", "description", "S&P ETF")
saveInstruments("MyInstruments2")
CompareInstrumentFiles("MyInstruments1", "MyInstruments2")
# Clean up
setwd(old.wd)
reloadInstruments(bak)

## End(Not run)
```

currencies

Currency symbols (like any symbol) may be any combination of alphanumeric characters, but the FX market has a convention that says that the first currency in a currency pair is the 'target' and the second currency in the symbol pair is the currency the rate ticks in. So 'EURUSD' can be read as 'USD per 1 EUR'.

exchange_rate

Constructor for spot exchange rate instruments

Description

Currency symbols (like any symbol) may be any combination of alphanumeric characters, but the FX market has a convention that says that the first currency in a currency pair is the 'target' and the second currency in the symbol pair is the currency the rate ticks in. So 'EURUSD' can be read as 'USD per 1 EUR'.
Usage

```python
exchange_rate(primary_id = NULL, currency = NULL, counter_currency = NULL,
tick_size = 0.01, identifiers = NULL, assign_i = TRUE,
overwrite = TRUE, ...)
```

Arguments

- `primary_id`: string identifier, usually expressed as a currency pair 'USDYEN' or 'EURGBP'
- `currency`: string identifying the currency the exchange rate ticks in
- `counter_currency`: string identifying the currency which the rate uses as the base 'per 1' multiplier
- `tick_size`: minimum price change
- `identifiers`: named list of any other identifiers that should also be stored for this instrument
- `assign_i`: TRUE/FALSE. Should the instrument be assigned in the .instrument environment? (Default TRUE)
- `overwrite`: TRUE by default. If FALSE, an error will be thrown if there is already an instrument defined with the same primary_id.
- `...`: any other passthru parameters

Details

In FinancialInstrument the currency of the instrument should be the currency that the spot rate ticks in, so it will typically be the second currency listed in the symbol.

Thanks to Garrett See for helping sort out the inconsistencies in different naming and calculating conventions.

References


---

`expires`  
extract the correct expires value from an instrument

Description

Currently, there are methods for instrument, spread, character, and xts

Usage

```
expires(x, ...)
```

Arguments

- `x`: instrument or name of instrument
- `...`: arguments to be passed to methods
Details

Will return either the last expiration date before a given date, or the first expiration date after a given date (if expired==FALSE).

If an instrument contains a value for expires that does not include a day (e.g. "2012-03"), or if the expires value is estimated from a future_series primary_id, it will be assumed that the instrument expires on the first of the month (i.e. if the expires value of an instrument were "2012-03", or if there were no expires value but the suffix_id were "H12", the value returned would be "2012-03-01"). Note that most non-energy future_series expire after the first of the month indicated by their suffix_id and most energy products expire in the month prior to their suffix_id month.

Value

an expiration Date

Author(s)

Garrett See

See Also

expires.instrument, expires.character, sort_ids

getInstrument and buildHierarchy to see actual values stored in instrument

Examples

## Not run:
instr <- instrument("FOO_U1", currency=currency("USD"), multiplier=1,
        expires=c("2001-09-01", "2011-09-01", "2021-09-01"),
        assign_i=FALSE)

#last value of expires that's not after Sys.Date
expires(instr)

# First value of expires that hasn't already passed.
expires(instr, expired=FALSE)

# last value that's not after 2011-01-01
expires(instr, Date="2011-01-01")

# first value that's not before 2011-01-01
expires(instr, Date="2011-01-01", expired=FALSE)

## expires.character

expires("FOO_U1") # warning that FOO_U1 is not defined

instrument("FOO_U1", currency=currency("USD"), multiplier=1,
        expires=c("2001-09-01", "2011-09-01", "2021-09-01"),
        assign_i=TRUE)

expires("FOO_U1")

## End(Not run)
find.instrument

Find the primary_ids of instruments that contain certain strings

Description

Uses regular expression matching to find instruments

Usage

find.instrument(text, where = "anywhere", Symbols = ls_instruments(),
              ignore.case = TRUE, exclude = NULL, ...)

Arguments

text character string containing a regular expression. This is used by grep (see also) as the pattern argument.

where if “anywhere” all levels/attributes of the instruments will be searched. Otherwise, where can be used to specify in which levels/attributes to look. (e.g. c("name", "description") would only look for text in those 2 places.

Symbols the character ids of instruments to be searched. All are are searched by default.

ignore.case passed to grep; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

exclude character vector of names of levels/attributes that should not be searched.

... other arguments to pass through to grep

Value

character vector of primary_ids of instruments that contain the sought after text.

Author(s)

Garrett See

See Also

buildHierarchy, instrument.table.regex

Examples

## Not run:
instruments.bak <- as.list(FinancialInstrument:::instrument, all.names=TRUE)
rm_instruments(keep.currencies=FALSE)
currency("USD")
stock("SPY", "USD", description="S&P 500 ETF")
stock("DIA", "USD", description="DJIA ETF")
stock(c("AA", "AXP", "BA", "BAC", "CAT"), "USD", members.of='DJIA')
stock("BMW", currency("EUR"))
FindCommonInstrumentAttributes

Find attributes that more than one instrument have in common

Description
Find attributes that more than one instrument have in common

Usage
FindCommonInstrumentAttributes(Symbols, ...)

Arguments
Symbols character vector of primary_ids of instruments
... arguments to pass to getInstrument

Value
character vector of names of attributes that all Symbols' instruments have in common

Note
I really do not like the name of this function, so if it survives, its name may change

Author(s)
gsee
### fn_SpreadBuilder

**Calculate prices of a spread from 2 instruments.**

**Description**

Given 2 products, calculate spread values for as many columns as practicable.

**Usage**

```r
fn_SpreadBuilder(prod1, prod2, ratio = 1, currency = "USD", from = NULL, 
                  to = NULL, session_times = NULL, notional = TRUE, 
                  unique_method = c("make.index.unique", "duplicated", "least.liq", 
                                     "price.change"), silent = FALSE, auto.assign = TRUE, env = .GlobalEnv, 
                  ...)```

**Arguments**

- `prod1`: chr name of instrument that will be the 1st leg of a 2 leg spread (Can also be xts data for first product)
- `prod2`: chr name of instrument that will be the 2nd leg of a 2 leg spread (Can also be xts data for second product)
- `ratio`: Hedge ratio. Can be a single number, or a vector of same length as data.
- `currency`: chr name of currency denomination of the spread
- `from`: from Date to pass through to getSymbols if needed.
- `to`: to Date to pass through to getSymbols if needed.
- `session_times`: ISO-8601 time subset for the session time, in GMT, in the format 'T08:00/T14:59'
- `notional`: TRUE/FALSE. Should the prices be multiplied by contract multipliers before calculating the spread?
- `unique_method`: method for making the time series unique
- `silent`: silence warnings? (FALSE by default)

---

### Examples

```r
## Not run:
ibak <- as.list(FinancialInstrument:::instrument, all.names=TRUE)
Symbols <- c("SPY", "AAPL")
define_stocks(Symbols, addIBslot=FALSE)
update_instruments.SPDR("SPY")
update_instruments.TTR("AAPL", exchange="NASDAQ")
FindCommonInstrumentAttributes(Symbols)
FindCommonInstrumentAttributes(c(Symbols, "USD"))
reloadInstruments(ibak)

## End(Not run)
```
auto.assign If TRUE (the default) the constructed spread will be stored in symbol created with make_spread_id. Instrument metadata will also be created and stored with the same primary_id.

env If prod1 and prod2 are character, this is where to get the data. Also, if auto.assign is TRUE this is the environment in which to store the data (.GlobalEnv by default)

... other arguments to pass to getSymbols and/or make_spread_id

Details

prod1 and prod2 can be the names of instruments, or the xts objects themselves. Alternatively, prod2 can be omitted, and a vector of 2 instrument names can be given to prod1. See the last example for this usage.

If prod1 and prod2 are names (not xts data), it will try to get data for prod1 and prod2 from env (.GlobalEnv by default). If it cannot find the data, it will get it with a call to getSymbols. Prices are multiplied by multipliers and exchange rates to get notional values in the currency specified. The second leg’s notional values are multiplied by ratio. Then the difference is taken between the notional of leg1 and the new values for leg2.

‘make.index.unique’ uses the xts function make.index.unique ‘least.liq’ subsets the spread time series, by using the timestamps of the leg that has the fewest rows. ‘duplicated’ removes any duplicate indexes. ‘price.change’ only return rows where there was a price change in the Bid, Mid or Ask Price of the spread.

Value

an xts object with Bid, Ask, Mid columns, or Open, Close, Adjusted columns, or Open, Close columns, or Price column.

Note

requires quantmod

Author(s)

Lance Levenson, Brian Peterson, Garrett See

See Also

buildSpread synthetic.instrument formatSpreadPrice buildRatio

Examples

```r
## Not run:
currency("USD")
stock("SPY", "USD")
stock("DIA", "USD")
getSymbols(c("SPY","DIA"))

#can call with names of instrument/xts objects
fSB <- fn_SpreadBuilder("SPY","DIA")
```
**formatSpreadPrice**  

format the price of a synthetic instrument

---

**Description**

Divides the notional spread price by the spread multiplier and rounds prices to the nearest tick_size.

**Usage**

formatSpreadPrice(x, multiplier = 1, tick_size = 0.01)

**Arguments**

- **x**: xts price series
- **multiplier**: numeric multiplier (e.g. 1000 for crack spread to get from $ to $/bbl)
- **tick_size**: minimum price change of the spread

**Value**

price series of same length as x

**Author(s)**

Garrett See

**See Also**

buildSpread, fn_SpreadBuilder
Description

convert the primary_id or suffix_id of an instrument to a different format. Primarily intended for future_series instruments.

Usage

format_id(id, format = NULL, parse = c("id", "suffix"), sep = "_", ...)

Arguments

id character. the id to be reformatted. Can be either a primary_id or a suffix_id
format character string indicating how the id should be formatted. See Details.
parse character name of parsing method to use: "id" or "suffix"
sep character that will separate root_id and suffix_id of output if calling with parse="id"
... parameters to pass to the parsing function

Details

Formats for the suffix_id include 'CY', 'CYY', and 'CYYYY' where C is the month code and Y is numeric. 'MMMY', 'MMMYY', 'MMMYYYY' where MMM is an uppercase month abbreviation. '1xCY', '1xCYY', '1xCYYYY' for single-stock-futures.

There are currently only 2 formats available for option_series: 'opt2' and 'opt4' where opt2 uses a 2 digit year and opt4 uses a 4 digit year.

Value

character id of the appropriate format

Author(s)

Garrett See

See Also

parse_id, parse_suffix, M2C, month_cycle2numeric
**future_series**

**Constructors for series contracts**

**Description**

Constructors for series contracts on instruments such as options and futures

**Usage**

```
future_series(primary_id, root_id = NULL, suffix_id = NULL,
               first_traded = NULL, expires = NULL, identifiers = NULL,
               assign_i = TRUE, overwrite = TRUE, ...)
```

```
option_series(primary_id, root_id = NULL, suffix_id = NULL,
               first_traded = NULL, expires = NULL, callput = c("call", "put"),
               strike = NULL, identifiers = NULL, assign_i = TRUE, overwrite = TRUE,
               ...)
```

```
bond_series(primary_id, suffix_id, ..., first_traded = NULL,
             maturity = NULL, identifiers = NULL, payment_schedule = NULL,
             assign_i = TRUE)
```

**Arguments**

- **primary_id**: String describing the unique ID for the instrument. May be a vector for `future_series` and `option_series`.
- **root_id**: String product code or underlying_id, usually something like 'ES' or 'CL' for futures, or the underlying stock symbol (maybe preceded with a dot) for equity options.
- **suffix_id**: String suffix that should be associated with the series, usually something like 'Z9' or 'Mar10' denoting expiration and year.
- **first_traded**: String coercible to Date for first trading day.
- **expires**: String coercible to Date for expiration date.
- **identifiers**: Named list of any other identifiers that should also be stored for this instrument.
- **assign_i**: TRUE/FALSE. Should the instrument be assigned in the `.instrument` environment?
overwrite TRUE/FALSE. If FALSE, only first_traded and expires will be updated.
...
any other passthru parameters
callput Right of option; call or put
strike Strike price of option
maturity String coercible to Date for maturity date of bond series.
payment_schedule Not currently being implemented

Details
The root instrument (e.g. the future or option) must be defined first.
In custom parameters for these series contracts, we have often found it useful to store attributes such as local roll-on and roll-off dates (rolling not on the first_listed or expires).
For future_series and option_series you may either provide a primary_id (or vector of primary_ids), OR both a root_id and suffix_id.
Note that the code for bond and bond_series has not been updated recently and may not support all the features supported for option_series and future_series. Patches welcome.

Examples
```r
# Not run:
currency("USD")
future("ES","USD",multiplier=50, tick_size=0.25)
future_series("ES_U1")
future_series(root_id='ES',suffix_id='Z11')
stock('SPY','USD')
option('.SPY','USD',multiplier=100,underlying_id='SPY')
#can use either .SPY or SPY for the root_id.
#it will find the one that is option specs.
option_series('SPY_110917C125', expires='2011-09-16')
option_series(root_id='SPY',suffix_id='111022P125')
option_series(root_id='.SPY',suffix_id='111119C130')
#multiple series instruments at once.
future_series(c("ES_M12","ES_M12"))
option_series(c("SPY_110917C115","SPY_110917P115"))
# End(Not run)
```

getInstrument  
*Primary accessor function for getting objects of class 'instrument'*

Description
This function will search the .instrument environment for objects of class type, using first the primary_id and then any identifiers to locate the instrument. Finally, it will try adding 1 and then 2 dots to the beginning of the primary_id to see if an instrument was stored there to avoid naming conflicts.
Usage

getInstrument(x, Dates = NULL, silent = FALSE, type = "instrument")

Arguments

x          String identifier of instrument to retrieve
Dates      date range to retrieve 'as of', may not currently be implemented
silent     if TRUE, will not warn on failure, default FALSE
type       class of object to look for. See Details

Details

future and option objects may have a primary_id that begins with 1 or 2 dots (in order to avoid
naming conflicts). For example, the root specs for options (or futures) on the stock with ticker "SPY"
may be stored with a primary_id of "SPY", ".SPY", or "...SPY". getInstrument will try using each
possible primary_id until it finds an instrument of the appropriate type

Examples

## Not run:
option('.VX', multiplier=100,
       underlying_id=future('.VX', multiplier=1000,
       underlying_id=synthetic('VIX', currency("USD"))))

getInstrument("VIX")
getInstrument("VX") #returns the future
getInstrument("VX",type='option')
getInstrument('.VX') #finds the option

## End(Not run)

---

getSymbols.FI

getSymbols method for loading data from split files

Description

This function should probably get folded back into getSymbols.rda in quantmod.

Usage

getSymbols.FI(Symbols, from =getOption("getSymbols.FI.from", "2010-01-01"),
              to = getOption("getSymbols.FI.to", Sys.Date()), ...,
              dir = getOption("getSymbols.FI.dir", ""),
              return.class = getOption("getSymbols.FI.return.class", "xts"),
              extension = getOption("getSymbols.FI.extension", "rda"),
              split_method = getOption("getSymbols.FI.split_method", c("days", "common")),
              use_identifier = getOption("getSymbols.FI.use_identifier", NA),
              ...)
date_format = getOption("getSymbols.FI.date_format"),
verbose = getOption("getSymbols.FI.verbose", TRUE),
days_to_omit = getOption("getSymbols.FI.days_to_omit", c("Saturday", "Sunday")), indexTZ = getOption("getSymbols.FI.indexTZ", NA))

Arguments

Symbols a character vector specifying the names of each symbol to be loaded from
Retrieve data no earlier than this date. Default ’2010-01-01’.
to Retrieve data through this date. Default Sys.Date().
... any other passthru parameters
dir if not specified in getSymbolLookup, directory string to use. default ""
return.class only "xts" is currently supported
extension file extension, default "rda"
split_method string specifying the method used to split the files, currently ‘days’ or ‘common’, see setSymbolLookup.FI
use_identifier optional. identifier used to construct the primary_id of the instrument. If you use this, you must have previously defined the instrument
date_format format as per the strptime, see Details
verbose TRUE/FALSE
days_to_omit character vector of names of weekdays that should not be loaded. Default is c("Saturday", "Sunday"). Use NULL to attempt to load data for all days of the week.
indexTZ valid TZ string. (e.g. “America/Chicago” or “America/New_York”) See indexTZ.

Details

Meant to be called internally by getSymbols.
The symbol lookup table will most likely be loaded by setSymbolLookup.FI.
If date_format is NULL (the Default), we will assume an ISO date as changed by make.names, for example, 2010-12-01 would be assumed to be a file containing 2010.12.01.
If indexTZ is provided, the data will be converted to that timezone.
If auto.assign is FALSE, Symbols should be of length 1. Otherwise, getSymbols will give you an error that says “must use auto.assign=TRUE for multiple Symbols requests” However, if you were to call getSymbols.FI directly (which is NOT recommended) with auto.assign=FALSE and more than one Symbol, a list would be returned.
Argument matching for this function is as follows. If the user provides a value for an argument, that value will be used. If the user did not provide a value for an argument, but there is a value for that argument for the given Symbol in the Symbol Lookup Table (see setSymbolLookup.FI), that value will be used. Otherwise, the formal defaults will be used.

See Also

cSaveSymbols.days instrument setSymbolLookup.FI loadInstruments getSymbols
instrument

instrument class constructors

Description

All 'currency' instruments must be defined before instruments of other types may be defined.

Usage

instrument(primary_id, ..., currency, multiplier, tick_size = NULL, identifiers = NULL, type = NULL, assign_i = FALSE, overwrite = TRUE)

stock(primary_id, currency = NULL, multiplier = 1, tick_size = 0.01, identifiers = NULL, assign_i = TRUE, overwrite = TRUE, ...)

fund(primary_id, currency = NULL, multiplier = 1, tick_size = 0.01, identifiers = NULL, assign_i = TRUE, overwrite = TRUE, ...)

future(primary_id, currency, multiplier, tick_size = NULL, identifiers = NULL, assign_i = TRUE, overwrite = TRUE, ..., underlying_id = NULL)

option(primary_id, currency, multiplier, tick_size = NULL, identifiers = NULL, assign_i = TRUE, overwrite = TRUE, ..., underlying_id = NULL)

currency(primary_id, identifiers = NULL, assign_i = TRUE, ...)

bond(primary_id, currency, multiplier, tick_size = NULL, identifiers = NULL, assign_i = TRUE, overwrite = TRUE, ...)

Arguments

primary_id String describing the unique ID for the instrument. Most of the wrappers allow this to be a vector.

... Any other passthru parameters, including

Examples

```r
## Not run:
globals("SPY", src='yahoo')
dir.create("tmpdata")
saveSymbols.common("SPY", base_dir="tmpdata")
rm("SPY")
globals("SPY", src='FI', dir="tmpdata", split_method='common')
unlink("tmpdata/SPY", recursive=TRUE)
## End(Not run)
```
currency    String describing the currency ID of an object of type currency
multiplier  Numeric multiplier to apply to the price in the instrument to get to notional value.
tick_size   The tick increment of the instrument price in it's trading venue, as numeric quantity (e.g. 1/8 is .125)
identifiers Named list of any other identifiers that should also be stored for this instrument
type        instrument type to be appended to the class definition, typically not set by user
assign_i    TRUE/FALSE. Should the instrument be assigned to the .instrument environment? Default is FALSE for instrument, TRUE for wrappers.
overwrite   TRUE/FALSE. Should existing instruments with the same primary_id be overwritten? Default is TRUE. If FALSE, an error will be thrown and the instrument will not be created.
underlying_id For derivatives, the identifier of the instrument that this one is derived from, may be NULL for cash settled instruments

Details

In ...you may pass any other arbitrary instrument fields that will be used to create 'custom' fields. S3 classes in R are basically lists with a class attribute. We use this to our advantage to allow us to set arbitrary fields.

identifiers should be a named list to specify other identifiers beyond the primary_id. Please note that whenever possible, these should still be unique. Perhaps Bloomberg, Reuters-X.RIC, CUSIP, etc. getInstrument will return the first (and only the first) match that it finds, starting with the primary_id, and then searching the primary_ids of all instruments for each of the identifiers. Note that when a large number of instruments are defined, it is faster to find instruments by primary_id than by identifiers because it looks for primary_ids first.

The primary_id will be coerced within reason to a valid R variable name by using make.names. We also remove any leading '1' digit (a simple workaround to account for issues with the Reuters API). If you are defining an instrument that is not a currency, with a primary_id that already belongs to a currency, a new primary_id will be create using make.names. For example, stock("USD", currency("USD")), would create a stock with a primary_id of “USD.1” instead of overwriting the currency.

Please use some care to choose your primary identifiers so that R won’t complain. If you have better regular expression code, we’d be happy to include it.

Identifiers will also try to be discovered as regular named arguments passed in via ... We currently match any of the following: "CUSIP", "SEDOL", "ISIN", "OSI", "Bloomberg", "Reuters", "X.RIC", "CQG", "TT", "Yahoo" Others may be specified using a named list of identifiers, as described above.

assign_i will use assign to place the constructed instrument class object into the .instrument environment. Most of the special type-specific constructors will use assign_i=TRUE internally. Calling with assign_i=FALSE, or not specifying it, will return an object and will not store it. Use this option ether to wrap calls to instrument prior to further processing (and presumably assignment) or to test your parameters before assignment.

If overwrite=FALSE is used, an error will be thrown if any primary_ids are already in use.

As of version 0.10.0, the .instrument environment is located at the top level of the package. i.e. .instrument.
future and option are used to define the contract specs of a series of instruments. The primary_id for these can begin with 1 or 2 dots if you need to avoid overwriting another instrument. For example, if you have a stock with ‘SPY’ as the primary_id, you could use ‘.SPY’ as the primary_id of the option specs, and ‘..SPY’ as the primary_id of the single stock future specs. (or vice versa)

You can (optionally) provide a src argument in which case, it will be used in a call to setSymbolLookup.

See Also

currency, exchange_rate, option_series, future_series, spread, load_instruments

instrument.auto

Create an instrument based on name alone

Description

Given a name, this function will attempt to create an instrument of the appropriate type.

Usage

instrument.auto(primary_id, currency = NULL, multiplier = 1,
    silent = FALSE, default_type = "unknown", root = NULL,
    assign_i = TRUE, ...)

Arguments

primary_id character primary identifier of instrument to be created

currency character name of currency that instrument will be denominated it. Default="USD"

multiplier numeric product multiplier

silent TRUE/FALSE. silence warnings?

default_type What type of instrument to make if it is not clear from the primary_id. ("stock", "future", etc.) Default is NULL.

root character string to pass to parse_id to be used as the root_id for easier/more accurate parsing.

assign_i TRUE/FALSE. Should the instrument be assigned in the .instrument environment?

... other passthrough parameters

Details

If currency is not already defined, it will be defined (unless it is not 3 uppercase characters). The default value for currency is “USD”. If you do not provide a value for currency, “USD” will be defined and used to create the instrument.

If primary_id is 6 uppercase letters and default_type is not provided, it will be assumed that it is the primary_id of an exchange_rate, in which case, the 1st and 2nd half of primary_id will be defined as currencys if not the names of already defined instruments. If the primary_id begins
with a “^” it will be assumed that it is a yahoo symbol and that the instrument is an index (synthetic), and the ‘src’ will be set to “yahoo”. (see setSymbolLookup)

If it is not clear from the primary_id what type of instrument to create, an instrument of type default_type will be created (which is NULL by default). This will happen when primary_id is that of a stock, future, option, or bond. This may also happen if primary_id is that of a future_series or option_series but the corresponding future or option cannot be found. In this case, the instrument type would be default_type, but a lot of things would be filled in as if it were a valid series instrument (e.g. ‘expires’, ‘strike’, ‘suffix_id’, etc.)

Value

Primarily called for its side-effect, but will return the name of the instrument that was created

Note

This is not intended to be used to create instruments of type stock, future, option, or bond although it may be updated in the future.

Author(s)

Garrett See

Examples

## Not run:
instrument.auto("CL_H1.U1")
getInstrument("CL_H1.U1") #guaranteed_spread

instrument.auto("ES_H1.YM_H1")
getInstrument("ES_H1.YM_H1") #synthetic

currency(c("USD","EUR"))
instrument.auto("EURUSD")
getInstrument("EURUSD") #made an exchange_rate

instrument.auto("VX_H11") #no root future defined yet!
getInstrument("VX_H11") #couldn’t find future, didn’t make future_series
future("VX","USD",1000,underlying_id=synthetic("SPX","USD")) #make the root
instrument.auto("VX_H11") #and try again
getInstrument("VX_H11") #made a future_series

## End(Not run)

---

**instrument.table**

Create data.frame with attributes of all instruments
**Description**

A wrapper for `buildHierarchy`, that defaults to returning all attributes. By default it looks for the instrument with the most attribute levels, and uses those attributes for columns. If you would prefer to use the attribute levels of a given instrument to build the columns, use `attrs.of`.

**Usage**

```r
instrument.table(symbols = NULL, exclude = NULL, attrs.of = NULL)
```

**Arguments**

- `symbols` A vector of instrument names to include
- `exclude` A vector of names of attributes that should not be included in the returned data.frame
- `attrs.of` name of a FinancialInstrument instrument. Returned data.frame columns will be the attributes of instrument.

**Details**

if there are some attributes that you do not want to be included in the returned data.frame, specify them with exclude.

**Value**

data.frame

**Author(s)**

Garrett See

**See Also**

`buildHierarchy`, `instrument`

**Examples**

```r
## Not run:
currency('USD')
stock('GM','USD',exchange='NYSE')
stock('XOM','USD',description='Exxon Mobil')
instrument.table()
#Usually, currencies will not have as many attribute levels
#as other instruments, so you may want to exclude them from the table.
it <- instrument.table(exclude='USD|GM', attrs.of = "XOM") #columns created based on XOM instrument
#it <- instrument.table(exclude=c('USD','GM'), attrs.of = "XOM") #same thing
it <- instrument.table(exclude='tick_size|description|exchange')
```

## End(Not run)
instrument_attr  Add or change an attribute of an instrument

Description

This function will add or overwrite the data stored in the specified slot of the specified instrument.

Usage

instrument_attr(primary_id, attr, value, ...)

Arguments

primary_id  primary_id of the instrument that will be updated
attr  Name of the slot that will be added or changed
value  What to assign to the attr slot of the primary_id instrument
...

arguments to pass to getInstrument. For example, type could be provided to allow for primary_id to be an identifier that is shared by more that one instrument (of different types)

Details

If the attr you are trying to change is the “primary_id,” the instrument will be renamed. (A copy of the instrument will be stored by the name of value and the old instrument will be removed.)

If the attr you are changing is “type”, the instrument will be reclassed with that type. If attr is “src”, value will be used in a call to setSymbolLookup. Other checks are in place to make sure that “currency” remains a currency object and that “multiplier” and “tick_size” can only be changed to reasonable values.

If attr is “identifiers” and value is NULL, identifiers will be set to list(). If value is not a list, add.identifier will be called with value. add.identifier will convert value to a list and append it to the current identifiers

Value

called for side-effect

Note

You can remove an attribute/level from an instrument by calling this function with value=NULL

Examples

```r
## Not run:
currency("USD")
stock("SPY","USD")
instrument_attr("USD","description","U.S. Dollar")
instrument_attr("SPY","description", "An ETF")
```
getInstrument("USD")
getInstrument("SPY")

# Call with value=NULL to remove an attribute
instrument_attr("SPY", "description", NULL)
getInstrument("SPY")

instrument_attr("SPY","primary_id","SPX") # move/rename it
instrument_attr("SPX","type","synthetic")  # re-class
instrument_attr("SPX","src",list(src='yahoo',name='GSPC'))  # setSymbolLookup
getSymbols("SPX")  # knows where to look because the last line setSymbolLookup
getInstrument("SPX")

## End(Not run)

---

**is.currency**

*class test for object supposedly of type 'currency'*

**Description**

class test for object supposedly of type 'currency'

**Usage**

is.currency(x)

**Arguments**

x            object to test for type

---

**is.currency.name**

*check each element of a character vector to see if it is either the primary_id or an identifier of a currency*

**Description**

check each element of a character vector to see if it is either the primary_id or an identifier of a currency

**Usage**

is.currency.name(x)

**Arguments**

x            character vector
is.instrument

*class test for object supposedly of type 'instrument'*

**Description**

class test for object supposedly of type 'instrument'

**Usage**

```r
is.instrument(x)
```

**Arguments**

- `x` object to test for type

---

is.instrument.name

*check each element of a character vector to see if it is either the primary_id or an identifier of an instrument*

**Description**

check each element of a character vector to see if it is either the primary_id or an identifier of an instrument

**Usage**

```r
is.instrument.name(x)
```

**Arguments**

- `x` character vector

**Value**

logical vector
load.instruments

load instrument metadata into the .instrument environment

Description

This function will load instrument metadata (data about the data) either from a file specified by the file argument or from a data.frame specified by the metadata argument.

Usage

load.instruments(file = NULL, ..., metadata = NULL, id_col = 1, default_type = "stock", identifier_cols = NULL, overwrite = TRUE)

Arguments

file string identifying file to load, default NULL, see Details
... any other passthrough parameters
metadata optional, data.frame containing metadata, default NULL, see Details
id_col numeric column containing id if primary_id isn’t defined, default 1
default_type character string to use as instrument type fallback, see Details
identifier_cols character vector of field names to be passed as identifiers, see Details
overwrite TRUE/FALSE. See instrument.

Details

The function will attempt to make reasonable assumptions about what you’re trying to do, but this isn’t magic.

You will typically need to specify the type of instrument to be loaded, failure to do so will generate a Warning and default_type will be used.

You will need to specify a primary_id, or define a id_col that contains the data to be used as the primary_id of the instrument.

You will need to specify a currency, unless the instrument type is ‘currency’

Use the identifier_cols argument to specify which fields (if any) in the CSV are to be passed to instrument as the identifiers argument

Typically, columns will exist for multiplier and tick_size.

Any other columns necessary to define the specified instrument type will also be required to avoid fatal Errors.

Additional columns will be processed, either as additional identifiers for recognized identifier names, or as custom fields. See instrument for more information on custom fields.

See Also

loadInstruments, instrument, setsymbollookup.FI, getSymbols, getSymbols.FI
Examples

```r
## Not run:
load.instruments(system.file('data/currencies.csv', package='FinancialInstrument'))
load.instruments(system.file('data/root_contracts.csv', package='FinancialInstrument'))
load.instruments(system.file('data/future_series.csv', package='FinancialInstrument'))
```

## End(Not run)

**ls_by_currency**  
shows or removes instruments of given currency denomination(s)

Description

`ls_` functions get names of instruments denominated in a given currency (or currencies) `rm_` functions remove instruments of a given currency

Usage

```r
ls_by_currency(currency, pattern = NULL, match = TRUE,  
show.currencies = FALSE)
```

```r
rm_by_currency(x, currency, keep.currencies = TRUE)
```

```r
ls_USD(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_AUD(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_GBP(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_CAD(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_EUR(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_JPY(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_CHF(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_HKD(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_SEK(pattern = NULL, match = TRUE, show.currencies = FALSE)
```

```r
ls_NZD(pattern = NULL, match = TRUE, show.currencies = FALSE)
```
Arguments

- **currency**: chr vector of names of currency
- **pattern**: an optional regular expression. Only names matching 'pattern' are returned.
- **match**: exact match?
- **show.currencies**: include names of currency instruments in the returned names?
- **x**: what to remove. chr vector.
- **keep.currencies**: Do not delete currency instruments when deleting multiple instruments.

Value

ls_ functions return vector of instrument names rm_ functions return invisible / called for side-effect.

Author(s)

Garrett See

See Also

ls_instruments, ls_currencies, rm_instruments, rm_currencies, twsInstrument, instrument

Examples

```r
# Not run:
# First create instruments
currency(c('USD','CAD','GBP'))
stock(c('CM','CNQ','CAD'))
stock(c('BET','BARC','GBP'))
stock(c('SPY','DIA','USD'))

# now the examples
ls_by_currency(c('CAD','GBP'))
ls_USD()
ls_CAD()

# 2 ways to remove all instruments of a currency
rm_instruments(ls_USD())
# rm_instruments(ls_GBP(), keep.currencies=FALSE)
rm_by_currency(,'CAD')
# rm_by_currency( , 'CAD', keep.currencies=FALSE)

## End(Not run)
```
**ls_by_expiry**

*list or remove instruments by expiration date*

**Description**

show names of or remove instruments that expire on a given date

**Usage**

`ls_by_expiry(expiry, pattern = NULL, match = TRUE)`

`rm_by_expiry(x, expiry)`

**Arguments**

expiry  expiration date that should correspond to the ‘expires’ field of an instrument
pattern  an optional regular expression. Only names matching ‘pattern’ are returned.
match  exact match of pattern?
x  what to remove

**Details**

`ls_by_expiry` will find instruments that have a field named either “expiry” or “expires” with a value that matches `expiry`.

**Value**

`ls_by_expiry` gives a vector of names of instruments that expire on the given expiry. `rm_by_expiry` is called for its side-effect.

**Author(s)**

Garrett See

**See Also**

`ls_instruments`, `ls_options`, `ls_calls`, `ls_puts`, `ls_futures`, `ls_derivatives`

**Examples**

```
## Not run:
ls_by_expiry('20110917')
ls_by_expiry('20110917',ls_options())

## End(Not run)
```
**Description**

show unique expiration dates of instruments

**Usage**

```r
ls_expiries(pattern = NULL, match = TRUE, underlying_id = NULL, type = "derivative")
```

```r
ls_expires(pattern = NULL, match = TRUE, underlying_id = NULL, type = "derivative")
```

**Arguments**

- `pattern` optional regular expression.
- `match` exact match?
- `underlying_id` chr name of underlying or vector of underlying_ids. If NULL, all underlyings will be used
- `type` chr string name of class that instruments to be returned must inherit.

**Details**

`ls_expires` is an alias. (plural of expires?)

type is currently only implemented for ‘derivative’, ‘future’, ‘option’, ‘call’ and ‘put’ internally, a call is made to the appropriate `ls_` function.

**Value**

named chr vector with length of unique expiration dates of derivatives of class `type` and having an underlying_id of `underlying_id` if given.

**Note**

This should be updated to deal with dates instead of character strings

**Author(s)**

Garrett

**See Also**

`ls_instruments_by` for things like e.g. `ls_instruments_by('expires','20110916'), ls_instruments, ls_derivatives, ls_options, ls_calls, buildHierarchy, instrument.table`
Examples

```r
## Not run:
option_series.yahoo('SPY')
option_series.yahoo('DIA',NULL)
ls_expiries()

## End(Not run)
```

### ls_instruments

*List or Remove instrument objects*

display the names of or delete instruments, stocks, options, futures, currencies, bonds, funds, spreads, guaranteed_spreads, synthetics, derivatives, or non-derivatives.

Usage

```r
ls_instruments(pattern = NULL, match = TRUE, verbose = TRUE)
ls_stocks(pattern = NULL, match = TRUE)
ls_options(pattern = NULL, match = TRUE, include.series = TRUE)
ls_option_series(pattern = NULL, match = TRUE)
ls_futures(pattern = NULL, match = TRUE, include.series = TRUE)
ls_future_series(pattern = NULL, match = TRUE)
ls_currencies(pattern = NULL, match = TRUE, includeFX = FALSE)
ls_non_currencies(pattern = NULL, match = TRUE, includeFX = TRUE)
ls_exchange_rates(pattern = NULL, match = TRUE)
ls_FX(pattern = NULL, match = TRUE)
ls_bonds(pattern = NULL, match = TRUE)
ls_funds(pattern = NULL, match = TRUE)
ls_spreads(pattern = NULL, match = TRUE)
ls_guaranteed_spreads(pattern = NULL, match = TRUE)
```
\textbf{ls\_instruments}

\begin{verbatim}
ls_synthetics(pattern = NULL, match = TRUE)
ls_ICS(pattern = NULL, match = TRUE)
ls_ICS_roots(pattern = NULL, match = TRUE)
ls_derivatives(pattern = NULL, match = TRUE)
ls_non_derivatives(pattern = NULL, match = TRUE)
ls_calls(pattern = NULL, match = TRUE)
ls_puts(pattern = NULL, match = TRUE)
rm_instruments(x, keep.currencies = TRUE)
rm_stocks(x)
rm_options(x)
rm_option_series(x)
rm_futures(x)
rm_future_series(x)
rm_currencies(x)
rm_exchange_rates(x)
rm_FX(x)
rm_bonds(x)
rm_funds(x)
rm_spreads(x)
rm_synthetics(x)
rm_derivatives(x)
rm_non_derivatives(x, keep.currencies = TRUE)
\end{verbatim}

\textbf{Arguments}

- \textbf{pattern} \hspace{1cm} an optional regular expression. Only names matching `pattern` are returned.
- \textbf{match} \hspace{1cm} return only exact matches?
verbose
be verbose?
include.series
should future.series or option.series instruments be included.
includeFX
should exchange.rates be included in ls_non_currencies results
x
what to remove. if not supplied all instruments of relevent class will be removed.
For ls_defined.by x is the string describing how the instrument was defined.
keep.currencies
If TRUE, currencies will not be deleted.

Details

ls functions return the names of all the instruments of the class implied by the function name. rm functions remove the instruments of the class implied by the function name
rm_instruments and rm_non_derivatives will not delete currencies unless the keep.currencies argument is FALSE.
For the rm functions, x can be a vector of instrument names, or nothing. If x is missing, all instruments of the relevant type will be removed.
It can be useful to nest these functions to get things like futures denominated in USD.

Value

ls functions return vector of character strings corresponding to instruments of requested type rm functions are called for side-effect

Author(s)

Garrett See

See Also

ls_instruments_by, ls_by_currency, ls_by_expiry, ls, rm, instrument, stock, future, option, currency, FinancialInstrument::sort_ids

Examples

## Not run:
#rm_instruments(keep.currencies=FALSE) #remove everything from .instrument

# First, create some instruments
currency(c("USD", "EUR", "JPY"))
#stocks
stock(c("S", "SE", "SEE", "SPY"), 'USD')
synthetic("SPX", "USD", src=list(src='yahoo', name='^GSPC'))
#derivatives
option(."SPY", 'USD', multiplier=100, underlying_id='SPY')
option_series(root_id="SPY", expires='2011-06-18', callput='put', strike=130)
option_series(root_id="SPY", expires='2011-09-17', callput='put', strike=130)
option_series(root_id="SPY", expires='2011-06-18', callput='call', strike=130)
future('ES', 'USD', multiplier=50, expires='2011-09-16', underlying_id='SPX')
option('.ES', 'USD', multiplier=1, expires='2011-06', strike=1350, right='C', underlying_id='ES')
# Now, the examples
ls_instruments() #all instruments
ls_instruments("SE") #only the one stock
ls_instruments("S", match=FALSE) #anything with "S" in name

ls_currencies()
ls_stocks()
ls_options()
ls_futures()
ls_derivatives()
ls_puts()
ls_non_derivatives()
#ls_by_expiry('20110618',ls_puts()) #put options that expire on Jun 18th, 2011
#ls_puts(ls_by_expiry('20110618')) #same thing

rm_options('SPY_110618C130')
rm_futures()
ls_instruments()
rm_instruments('EUR') #Incorrect
rm_instruments('EUR', keep.currencies=FALSE) #remove the currency
rm_currencies('JPY') #or remove currency like this
ls_currencies()
ls_instruments()

rm_instruments() #remove all but currencies
rm_currencies()

option_series.yahoo('DIA')
ls_instruments_by('underlying_id','DIA') #underlying_id must exactly match 'DIA'
ls_derivatives('DIA',match=FALSE) #primary_ids that contain 'DIA'
rm_instruments()

## End(Not run)

### ls_instruments_by

**Subset names of instruments**

**Description**

list names of instruments that have an attribute that matches some value

**Usage**

ls_instruments_by(what, value, in.slot = NULL, pattern = NULL, match = TRUE)
Arguments

what  What attribute? (e.g. “currency”, “type”, “strike”, etc.)
value What value must the attribute have? (e.g. “EUR”, “option”, 100, etc.). If missing or NULL, the names of all instruments that have a what slot will be returned
in.slot If the attribute you are looking for is stored inside another slot, this is the name of that slot. (usually "IB")
pattern only return instruments with pattern in the name
match should pattern match names exactly?

Details

list instruments that have a given attribute level with a given value.

Value

chr vector of instrument names

Author(s)

Garrett See

See Also

buildHierarchy, instrument.table, ls_instruments

Examples

## Not run:
stock(c("GOOG","INTC"),currency("USD"))
synthetic("SnP","USD",src=list(name='^GSPC',src='yahoo'))
ls_instruments_by('type','stock')
ls_instruments_by("name",NULL,in.slot='src')
ls_instruments_by('src',NULL)

## End(Not run)

---

ls_strikes  

show strike prices of defined options

Description

list the strike prices of previously defined options.

Usage

ls_strikes(pattern = NULL)
Arguments

pattern an optional regular expression. Only names matching 'pattern' are returned.

Details

If no option names are supplied, the strike prices of all defined options will be returned.

Value

vector of strike prices

Author(s)

Garrett See

See Also

ls_options, ls_calls, ls_puts ls_instruments_by ls_underlyings

Examples

## Not run:
option_series.yahoo('SPY')
ls_strikes(ls_options('SPY'))
## End(Not run)

---

**ls_underlyings**  
show names of underlyings

Description

shows names that are stored in the underlying_id slot of derivative instruments

Usage

ls_underlyings(pattern = NULL, match = TRUE)

Arguments

pattern an optional regular expression. Only names matching 'pattern' are returned.

match require exact match?

Details

first calls ls_derivatives, then looks for unique underlying_ids. If no derivatives have been defined, nothing will be returned.
make_spread_id

Value
chr vector of names of unique underlying_ids

Author(s)
Garrett See

See Also
ls_instruments_by, ls_derivatives, ls_options, ls_futures

Examples
## Not run:
ls_underlyings()

## End(Not run)

make_spread_id Construct a primary_id for a spread instrument from the primary_ids of its members

Description
Construct a primary_id for a spread instrument from the primary_ids of its members

Usage
make_spread_id(x, root = NULL, format = NULL, sep = "_")

Arguments
x character vector of member primary_ids
root Optional character string of root_id to use.
format String indicating how to format the suffix_ids of the spread. If NULL (the default), or FALSE, no formatting will be done. See format_id for other accepted values for format
sep character string to separate root_id and suffix_id

Value
character string that can be used as a primary_id for a spread instrument

Author(s)
Garrett See


**month_cycle2numeric**

coease month_cycle2numeric to a numeric vector

**Description**

This will convert month codes or month names to numeric months.

**Usage**

```r
month_cycle2numeric(...)
```

**Arguments**

`...` the expiration months of a `future`. See examples.

**Details**

Input can be a vector, comma-delimited string, or multiple strings. All inputs should be similar. Do not mix month names, codes and numbers in the same call.

MC2N is an alias

**Value**

numeric vector

**Author(s)**

Garrett See

**See Also**

`M2C`, `C2M`, `next.future_id future`
Examples

MC2N("H,M,U,Z") # from single string
MC2N(c("H","M","U","Z")) # from single vector
MC2N("h", "M", "u", "Z") # from multiple strings
MC2N(c("F","G","H", c("X","Z"))) # from multiple vectors
month_cycle2numeric("Mar,jun,SEP,dec")
month_cycle2numeric("Mar", "jun", "SEP", "dec")
MC2N("March,june,sep,decem") #spaces between commas are ok
month_cycle2numeric("3,6,9,12")
month_cycle2numeric(seq(3,12,3))

next.future_id

Get the primary_id of the next-to-expire (previously expiring) future_series instrument

Description

Using parse_id, this will figure out where in the month_cycle that id belongs. Then, it will use the next (previous) month in month_cycle to construct the id of the next-to-expire contract.

Usage

next.future_id(id, month_cycle = seq(3, 12, 3), root = NULL, format = NULL)
prev.future_id(id, month_cycle = seq(3, 12, 3), root = NULL, format = NULL)

Arguments

id character string primary_id of a future_series instrument
month_cycle months in which contracts expire. numeric or month codes. See Details.
root root_id. usually only used if there is no underscore in the id. See Details.
format how you would like the returned id to be formatted. If NULL, it will match the format of id. See Details.

Details

month_cycle can be a numeric vector (corresponding to the months in which contracts expire), or it can be a vector of month codes, a vector of month abbreviations, or a comma-delimited string of month codes or abbreviations, in which case an attempt will be made to convert it to a numeric vector. by passing it through month_cycle2numeric

root is primarily used when you have an id that does not have an underscore, in which case, providing root will make splitting the id into primary_id and suffix_id easier and more accurate. root can also be used if you want the returned id to be on a different future than the id you passed in (when used this way, format should also be used).
By default, (when called with `format=NULL`) the returned id will be of the same format as the id that was passed in. The format of the returned id can be specified with the `format` argument. See `format_id` for supported values of `format`.

**Value**

character

**Author(s)**

Garrett See

**See Also**

`format_id` for supported values of `format`. `month_cycle2numeric`

**Examples**

```r
next.future_id("ES,Z1","H,M,U,Z", format=NULL)
next.future_id("VIXAUG11", 1:12, root='VIX', format=NULL)
next.future_id("YM_Q11", seq(3,12,3)) #gives a warning about 'Q' not being part of month_cycle
```

---

**Notionalize**  
*Convert price series to/from notional value*

**Description**

Notionalize multiplies all prices by the contract multiplier. Denotionalize divides all prices by the contract multiplier.

**Usage**

```r
Notionalize(x, name, env = .GlobalEnv)
Denotionalize(x, name, env = .GlobalEnv)
```

**Arguments**

- `x` an xts object, or an object that is coercible to xts
- `name` primary_id of the instrument that has the multiplier; usually the same as the name of `x`
- `env` environment. where to find `x` if only its name is provided

**Details**

The multiplier is only applied to columns with prices. A column is considered to be a price column if its name contains “Open”, “High”, “Low”, “Close”, “Bid”, “Ask”, “Trade”, “Mid”, or “Price” and does not contain “Size”, “Sz”, “Volume”, “Qty”, “Quantity”, “OpInt”, “OpenInterest” (not case-sensitive)
option_series.yahoo
constructor for series of options using yahoo data

Description
Defines a chain or several chains of options by looking up necessary info from yahoo.

Usage
```r
option_series.yahoo(symbol, Exp, currency = "USD", multiplier = 100,
                      first_traded = NULL, tick_size = NULL, overwrite = TRUE)
```

Arguments
- **symbol**: character vector of ticker symbols of the underlying instruments (Currently, should only be stock tickers)
- **Exp**: Expiration date or dates to be passed to getOptionChain
- **currency**: currency of underlying and options
- **multiplier**: contract multiplier. Usually 100 for stock options
- **first_traded**: first date that contracts are tradeable. Probably not applicable if defining several chains.
- **tick_size**: minimum price change of options.
- **overwrite**: if an instrument already exists, should it be overwritten?

Details
If **Exp** is missing it will define only the nearby options. If **Exp** is NULL it will define all options
If **first_traded** and/or **tick_size** should not be the same for all options being defined, they should be left NULL and defined outside of this function.
parse_id

Value
Called for side-effect. The instrument that is created and stored will inherit option_series, option, and instrument classes.

Note
Has only been tested with stock options. The options’ currency should be the same as the underlying’s.

Author(s)
Garrett See

References
Yahoo http://finance.yahoo.com

See Also
option_series, option, instrument, getOptionChain

Examples
## Not run:
option_series.yahoo('SPY') #only nearby calls and puts
option_series.yahoo('DIA', Exp=NULL) #all chains
ls_instruments()
## End(Not run)

---

parse_id Parse a primary_id

Description
Extract/infer descriptive information about an instrument from its name.

Usage
parse_id(x, silent = TRUE, root = NULL)

Arguments
x the id to be parsed (e.g. ‘ES_U11’, ‘SPY_111217C130’)
silent silence warnings?
root character name of instrument root_id. Optionally provide this to make parsing easier.
Details

This function is primarily intended to be used on the names of future_series and option_series instruments, and it will work best if the id has an underscore in it that separates the root_id from the suffix_id. (However, it should be able to handle most ids even if the underscore is missing). After splitting x into a root_id and suffix_id, the suffix_id is passed to parse_suffix (see also) for further processing.

TODO: add support for bond_series.

Value

a list of class ‘id.list’ containing ‘root’ and ‘suffix’ as well as what is returned from parse_suffix (type, month, year, strike, right, cm, cc, format)

Note

this function will identify x as an exchange_rate only if it is 6 characters long and made up of 2 previously defined currency instruments.

Author(s)

Garrett See

See Also

parse_suffix

Examples

parse_id("ES_Z11")
parse_id("CLZ1")
parse_id("SPY_111217C130")

Description

extract information from the suffix_id of an instrument

Usage

parse_suffix(x, silent = TRUE)

Arguments

x the suffix_id to be parsed
silent silence warnings? (warning will usually be about inferring a 4 digit year from a 1 or 2 digit year)
Details

These would be recognized as a Sep 2011 outright futures contract: U1, U11, SEP1, SEP11, U2011, SEP2011

These would be recognized as a call with a strike of 122.5 that expires Sep 17, 2011: 110917C122.5, 20110917C122.5, 110917C00122500, 20110917C00122500

These would be recognized as Sep 2011 single stock futures: 1CU1, 1CU11, 1CSEP11, 1DU1 (dividend protected)

These would be recognized as Adjusted futures: cm.30 (30 day constant maturity future), cc.OI (continuous contract rolled when Open Interest rolls), cc.Vol (continuous contract roll when Volumn rolls), cc.Exp.1 (continuous contract rolled 1 day before Expiration)

Synthetics and spreads:

SPY.DIA -> type == synthetic;
U1.Z1 or U11.Z11 -> type == "calendar", "spread"; month == 'SEP', year == 2011
U1.0302 -> type == "ICS", "spread"; month == 'SEP', year == 2011
110917C125.110917P125 -> type == option_spread, spread

Value

an object of class ‘suffix.list’ which is a list containing ‘type’ of instrument, ‘month’ of expiration, ‘year’ of expiration, ‘strike’ price of option, ‘right’ of option (“C” or “P”), ‘cm’ (maturity in days of a constant maturity contract), ‘cc’ (method for calculating a continuous contract), ‘format’ (string that indicates the format of the unparsed id).

Author(s)

Garrett See

See Also

parse_id, format_id

Examples

parse_suffix("U11")
parse_suffix("110917C125")

redenominate
Redenominate (change the base of) an instrument

Description

Redenominate (change the base of) an instrument
Usage

redenominate(x, new_base = "USD", old_base = NULL, EOD_time = "15:00:00",
env = .GlobalEnv, silent = FALSE)

Arguments

x can be either an xts object or the name of an instrument.
new_base change the denomination to this; usually a currency.
old_base what is the current denomination?
EOD_time If data need to be converted to daily, this is the time of day to take the observation.
env environment that holds the data
silent silence warnings?

Details

If old_base is not provided, x must be the name of an instrument (or an object with the name of a defined instrument) so that the currency attribute of the instrument can be used. Otherwise, old_base must be provided.

If you want to convert to JPY something that is denominated in EUR, you must have data for the EURJPY (or JPYEUR) exchange rate. If you don’t have data for EURJPY, but you do have data for EURUSD and USDJPY, you could redenominate to USD, then redenominate to EUR, but this function is not yet smart enough to do that for you.

See the help for buildRatio also.

Value

xts object, with as many columns as practicable, that represents the value of an instrument in a different currency (base).

Note

this does not yet define any instruments or assign anything.

Author(s)

Garrett See

See Also

buildRatio
Examples

```
## Not run:
require(quantmod)
EURUSD <- getSymbols("EURUSD=x", src='yahoo', auto.assign=FALSE)
GLD <- getSymbols("GLD", src='yahoo', auto.assign=FALSE)
GLD.EUR <- redenominate(GLD,"EUR","USD") # can call with xts object
currency("USD")
stock("GLD","USD")
GLD.EUR <- redenominate('GLD','EUR') # can also call with instrument name
```

# End(Not run)

root_contracts  
future metadata to be used by load.instruments

Description

future metadata to be used by load.instruments

saveInstruments  
Save and Load all instrument definitions

Description

Saves (loads) the .instrument environment to (from) disk.

Usage

```
saveInstruments(file_name = "MyInstruments", dir = ",", compress = "gzip")
loadInstruments(file_name = "MyInstruments", dir = ")
reloadInstruments(file_name = "MyInstruments", dir = ")
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>name of file. e.g. “MyInstruments.RData”. As an experimental feature, a list or environment can be passed to file_name.</td>
</tr>
<tr>
<td>dir</td>
<td>Directory of file (defaults to current working directory. ie. &quot;)</td>
</tr>
<tr>
<td>compress</td>
<td>argument passed to save, default is &quot;gzip&quot;</td>
</tr>
</tbody>
</table>
Details

After you have defined some instruments, you can use `saveInstruments` to save the entire `.instrument` environment to disk.

`loadInstruments` will read a file that contains instruments and add those instrument definitions to your `.instrument` environment. `reloadInstruments` will remove all instruments in the current `.instrument` environment before loading instruments from disk.

The `file_name` should have a file extension of “RData”, “rda”, “R”, or “txt”. If the `file_name` does not end with one of those, “.RData” will be appended to the `file_name`.

If the file extension is “R” or “txt”, `saveInstruments` will create a text file of R code that can be sourced to load instruments back into the `.instrument` environment.

Value

Called for side-effect

Author(s)

Garrett See

See Also

`save`, `load`, `load.instrument`, `define_stocks`, `define_futures`, `define_options` (option_series.yahoo)

Examples

```r
## Not run:
stock("SPY", currency("USD"), 1)
tmpdir <- tempdir()
saveInstruments("MyInstruments.RData", dir=tmpdir)
rm_instruments(keep.currencies=FALSE)
loadInstruments("MyInstruments.RData", dir=tmpdir)
# write .R file that can be sourced
saveInstruments("MyInstruments.R", dir=tmpdir)
rm_instruments(keep.currencies=FALSE)
loadInstruments("MyInstruments.R", dir=tmpdir)
#source(file=paste(tmpdir, "MyInstruments.R", sep="/")) # same
unlink(tmpdir, recursive=TRUE)

## End(Not run)
```
saveSymbols.days

Usage

saveSymbols.days(Symbols, base_dir = "", extension = "rda", 
                env = .GlobalEnv)

saveSymbols.common(Symbols, base_dir = "", extension = "rda", 
                   env = .GlobalEnv)

Arguments

Symbols character vector of names of objects to be saved
base_dir character. directory in which to store data.
extension file extension ("rda")
env environment that holds the data to be saved (.GlobalEnv by default)

Details

If they do not already exist, subdirectories will be created for each of the Symbols. saveSymbols.common will save a single 'rda' file for each of the Symbols in that symbol's subdirectory. saveSymbols.days will split the data up into days and save a separate 'rda' file for each day in that symbol's subdirectory.

Value
called for side-effect.

See Also

getSymbols.FI

Examples

## Not run:
getSymbols("SPY", src='yahoo')
dir.create("tmpdata")
saveSymbols.common("SPY", base_dir="tmpdata")
rm("SPY")
getSymbols("SPY", src='FI', dir="tmpdata", split_method='common')
unlink("tmpdata/SPY", recursive=TRUE)

## End(Not run)
setSymbolLookup.FI

set quantmod-style SymbolLookup for instruments

Description

This function exists to tell 

getSymbols

where to look for your repository of market data.

Usage

setSymbolLookup.FI(base_dir, Symbols, ..., split_method = c("days", "common"),
storage_method = "rda", use_identifier = "primary_id",
extension = "rda", src = "FI")

Arguments

base_dir string specifying the base directory where data is stored, see Details
Symbols character vector of names of instruments for which to setSymbolLookup
split_method string specifying the method files are split, currently ‘days’ or ‘common’, see Details
storage_method currently only ‘rda’, but we will eventually support ‘indexing’ at least, and maybe others
use_identifier string identifying which column should be use to construct the primary_id of the instrument, default 'primary_id'
extension file extension, default "rda"
src which 

getSymbols

sub-type to use, default 

getSymbols.FI

by setting 'FI'

... any other passthru parameters

Details

The base_dir parameter must be set or the function will fail. This will vary by your local environment and operating system. For mixed-OS environments, we recommend doing some OS-detection and setting the network share to your data to a common location by operating system. For example, all Windows machines may use “M:/” and all *nix-style (linux, Mac) machines may use “/mnt/mktdata/”.

The split_method currently allows either ‘days’ or ‘common’, and expects the file or files to be in sub-directories named for the symbol. In high frequency data, it is standard practice to split the data by days, which is why that option is the default.

See Also

getSymbols.FI, instrument_attr, load.instruments, loadInstruments, setSymbolLookup
sort_ids

sort primary_ids of instruments

Description

Primarily intended for use on the primary_ids of future_series instruments. This will sort ids by expiration. All ids that do not contain month and year information will be sorted alphabetically (separately) and appended to the end of the other sorted ids.

Usage

sort_ids(ids, ...)

Arguments

ids character vector of ids

... arguments to pass through to parse_id

Details

If an instrument is defined, and has a date in its ‘expires’ field, that date will be used as the expiration date. Otherwise, it is assumed that the contract expires on the first day of its expiration month. This means that if some products are defined and other products that expire in the same month are not defined, the ones that are not defined will come first in the vector of sorted ids.

Value

sorted character vector of the same length as ids

Author(s)

Garrett See

See Also

parse_id

Examples

```r
## Not run:
ids <- c("ES_U11", "GLD", "SPY", "YM_Jun11", "DIA", "VX_V10")
sort_ids(ids)

## End(Not run)
```
**Description**

define spreads, guaranteed_spreads, butterflies, and other synthetic instruments

**Usage**

```r
synthetic(primary_id = NULL, currency = NULL, multiplier = 1,
          identifiers = NULL, assign_i = TRUE, overwrite = TRUE, ...,
          members = NULL, type = "synthetic")

synthetic.instrument(primary_id, currency, members, memberratio, ...,
                      multiplier = 1, tick_size = NULL, identifiers = NULL, assign_i = TRUE,
                      type = c("synthetic.instrument", "synthetic"))

spread(primary_id = NULL, currency = NULL, members, memberratio,
       tick_size = NULL, ..., multiplier = 1, identifiers = NULL,
       assign_i = TRUE)

butterfly(primary_id = NULL, currency = NULL, members, tick_size = NULL,
          identifiers = NULL, assign_i = TRUE, ...)

guaranteed_spread(primary_id = NULL, currency = NULL, root_id = NULL,
                   suffix_id = NULL, members = NULL, memberratio = c(1, -1), ...,
                   multiplier = NULL, identifiers = NULL, assign_i = TRUE,
                   tick_size = NULL)

ICS_root(primary_id, currency = NULL, members, multiplier = NULL,
         identifiers = NULL, assign_i = TRUE, overwrite = TRUE,
         tick_size = NULL, ...)

ICS(primary_id, assign_i = TRUE, identifiers = NULL, ...)
```

**Arguments**

- **primary_id** chr string of primary identifier of instrument to be defined.
- **currency** chr string name of currency denomination
- **multiplier** multiplier of the spread (1 / divisor for price weighted baskets)
- **identifiers** identifiers
- **assign_i** TRUE/FALSE. Should the instrument be assigned in the .instrument environment?
- **overwrite** if FALSE and an instrument with the same primary_id is already defined, an error will be thrown and no instruments will be created.
members vector of primary_ids of member instruments

type type of instrument; wrappers do not require this.

memberratio vector of weights for each leg. negative numbers for selling.

tick_size minimum price change of the spread

root_id instrument identifier for the root contract, default NULL

suffix_id identifiers for the member contract suffixes, default NULL, will be split as members, see Details

... any other passthrough parameters

Details

Simple derivatives like option or future contracts typically have one underlying instrument. While properties like strike and expiration vary for these derivative contracts or series, the underlying is well understood.

More complex derivatives are typically modeled as baskets of underlying products, and are typically traded over-the-counter or as proprietary in-house products.

The general synthetic function is intended to be extended to support these arbitrary baskets of assets.

spread guaranteed_spread and butterfly are wrappers for synthetic.instrument. synthetic.instrument will make a call to synthetic to create the final instrument.

The suffix_id parameter of wrapper functions such as guaranteed_spread is presumed to be a string describing the members. It will be strsplit using the regex "[\-;\:\_,\.]" to create the members vector, and potentially combined with a root_id.

Most wrappers will build primary_id if it is NULL, either by combining root_id and suffix_id, or by passing members in a call to make_spread_id.

ICS will build an Intercommodity Spread. Although the expiration date and ratio may change, the members of a given ICS will not change. Therefore, ICS_root can be used to hold the members of an Intercommodity Spread. If an ICS_root has not been defined, then members will be a required argument for ICS.

We welcome assistance from others to model more complex OTC derivatives such as swap products.

Value
called for side effect. stores an instrument in .instrument environment

Author(s)
Brian Peterson, Garrett See

See Also
instrument, future, option_series.yahoo
Examples

```r
## Not run:
stock('SPY', 'USD', 1)
stock('DIA', 'USD', 1)
spread('SPY', 'DIA', c('USD', c('SPY', 'DIA'), c(1, -1)))
## End(Not run)
```

to_secBATV  

Convert tick data to one-second data

Description

This is like taking a snapshot of the market at the end of every second, except the volume over the second is summed.

Usage

to_secBATV(x)

alltick2sec(getdir = "~/TRTH/tick/", savedir = "~/TRTH/sec/",
Symbols = list.files(getdir), overwrite = FALSE)

Arguments

- **x**  
  the xts series to convert to 1 minute BATV

- **getdir**  
  Directory that contains tick data

- **savedir**  
  Directory in which to save converted data

- **Symbols**  
  String names of instruments to convert

- **overwrite**  
  TRUE/FALSE. If file already exists in savedir, should it be overwritten?

Details


The primary purpose of these functions is to reduce the amount of data on disk so that it will take less time to load the data into memory.

If there are no trades or bid/ask price updates in a given second, we will not make a row for that timestamp. If there were no trades, but the bid or ask price changed, then we _will_ have a row but the Volume and Trade.Price will be NA.

If there are multiple trades in the same second, Volume will be the sum of the volume, but only the last trade price in that second will be printed. Similarly, if there is a trade, and then later in the same second, there is a bid/ask update, the last Bid/Ask Price/Size will be used.

alltick2sec is used to convert the data of several files from tick to one second frequency data.
**Value**

to_secBATV returns an xts object of one second frequency. alltick2sec returns a list of files that were converted.

**Note**

to_secBATV is used by the TRTH_BackFill.R script in the inst/parser directory of the FinancialInstrument package. These functions are specific to data created by that script and are not intended for more general use.

**Author(s)**

gsee

**Examples**

```r
## Not run:
getSymbols("CLU")
system.time(xsec <- to_secBATV(CLU))
convert.log <- alltick2sec()

## End(Not run)
```

---

**update_instruments.instrument**

*Update instruments with metadata from another instrument.*

**Description**

Update instruments with metadata from another instrument.

**Usage**

```r
update_instruments.instrument(Symbol, source_id, create.new = FALSE, ignore = "identifiers", assign.i = TRUE)
```

**Arguments**

- **Symbols**: character vector of primary_ids or other instrument identifiers. of instruments to be updated. Alternatively, Symbols can be an instrument or list of instruments.
- **source_id**: The primary_id (or other identifier) of an instrument, or an instrument. The source_id instrument will be used to update the metadata of Symbols’ instruments.
- **create.new**: If FALSE (Default), only attributes that exist but have empty values will be updated. If TRUE, new attributes will be created if source_id has them, but the Symbols do not.
ignore vector of names of instrument attributes that should not be copied to the updated instruments.

assign_i TRUE/FALSE. If TRUE, the updated instruments will be assigned back into the instrument environment. If FALSE, a list of updated instruments will be returned.

Details

By default, only attributes that have a value of "" will be given a new value.

If create.new is TRUE, then if there are attributes in source_id that are not in the Symbols’ instrument, those attributes will be copied to the updated instruments unless they are in ignore.

Value

if isTRUE(assign_i) a vector of primary_ids of the instruments that were updated. Otherwise, a list of updated instrument objects.

Note

one way to overwrite attributes of one instrument with those of another is to first set equal to "" those attributes that you want to overwrite, then use update_instruments.instrument to copy the attributes.

Author(s)

Garrett See

See Also

update_instruments.yahoo, all.equal.instrument

Examples

## Not run:
#rm_instruments()
currency("USD")
synthetic("SPX", "USD", identifiers=list(yahoo="GSPC"),
  tick_size=0.01,
  liquidHours="08:30:00/15:00:00",
  extraField='something else',
  assign_i=TRUE)
stock("SPY", "USD", liquidHours="", assign_i=TRUE)
all.equal(getInstrument("SPX"), getInstrument("SPY"))
getInstrument("SPY")
## update SPY metadata based on the metadata of SPX
## Only attributes that == "" are updated by default
update_instruments.instrument("SPY", "SPX", assign_i=FALSE) #liquidHours
update_instruments.instrument("SPY", "SPX", create.new=TRUE,
  ignore=("identifiers", "type"),
  assign_i=FALSE)

# Although you probably do NOT want to, this will
# copy everything new -- including identifiers and type!
update_instruments.instrument("SPY", "S PX", create.new=TRUE, ignore=NULL,
assign_i=FALSE)

## End(Not run)

---

**update_instruments.iShares**  
*update iShares and SPDR ETF metadata*

### Description
This will update previously defined iShares or SPDR ETF instruments. Both functions will add attributes for “Name”, and “FundFamily” (“iShares” or “SPDR”). `update_instruments.iShares` will also add an attribute for “MgmtFees”

### Usage

```r
update_instruments.iShares(Symbols, silent = FALSE)
update_instruments.SPDR(Symbols, silent = FALSE)
```

### Arguments
- **Symbols** character vector of iShares ETF ticker symbols. If not specified, `unique(c(ls_funds(), ls_stocks()))` will be used.
- **silent** silence the warning that no iShares are defined?

### Value
called for side-effect

### Note
`update_instruments.SPDR` will probably NOT work on Windows because in the call to `download.file` it uses method=curl since it has to download from an https URL scheme.

### Author(s)
Garrett See

### References


### See Also

- `update_instruments.yahoo`, `update_instruments.TTR`, `twsInstrument:::update_instruments.IB`, `update_instruments.instrument`, `update_instruments.morningstar`, `update_instruments.masterDATA`
Examples

```r
## Not run:
stock("IWC", currency("USD"))
update_instruments.iShares("IWC")
getInstrument("IWC")

Symbols <- stock(c("SPY", "JNK"), currency("USD"))
update_instruments.SPDR(Symbols)
buildHierarchy(c("SPY", "JNK"), "Name")

## End(Not run)
```

update_instruments.masterDATA

*Update instrument metadata for ETFs*

Description

Uses the masterDATA.com list of ETFs and ETNs to update previously defined instruments.

Usage

```r
update_instruments.masterDATA(Symbols, silent = FALSE)
```

```r
update_instruments.md(Symbols, silent = FALSE)
```

Arguments

- **Symbols**: character vector of Symbols of ETFs
- **silent**: silence warnings?

Details

update_instruments.md is an alias.

MasterDATA classifies each ETF into one of six Fund.Types. From their website:

- **US Equity ETF**: All constituents trade on a US exchange. Both ProShares and Rydex sponsor ETFs with the objective of achieving the performance (or a multiple of the performance) of several major US stock indexes. These ETFs currently are included in this category despite the fact that their constituent lists are generally not limited to US stocks.
- **Global Equity ETF**: One or more of the constituents do not trade on a US Exchange.
- **Fixed Income ETF**: The constituent list contains government and / or corporate debt instruments. ETFs with this classification will not be considered for inclusion in MasterDATA’s index / ETF compilation list.
- **Commodity Based ETF**: This classification of ETF has no constituents but is structured to reflect the valuation of a commodity such as gold, silver, oil or interest rates. ETFs with this classification will not be considered for inclusion in MasterDATA’s index / ETF compilation list.
Exchange Traded Notes: A type of unsecured, unsubordinated debt security that was first issued by Barclays Bank PLC. The purpose of ETNs is to create a type of security that combines both the aspects of bonds and exchange traded funds (ETF). Similar to ETFs, ETNs are traded on a major exchange.

Value

called for side-effect. Each ETF that is updated will be given instrument attributes of “Name” and “Fund.Type”

Author(s)

Garrett See

References

http://masterDATA.com (http://www.masterdata.com/helpfiles/ETF_List_Downloads/AllTypes.csv)

See Also

update_instruments.yahoo, update_instruments.instrument

Examples

```r
## Not run:
stock(s <- c("SPY", "DIA"), currency("USD"))
update_instruments.masterDATA(s)
buildHierarchy(s, "Name", "Fund.Type", "defined.by")

## End(Not run)
```

update_instruments.morningstar

Update instrument metadata for ETFs

Description

Currently, this only updates ETFs. It will add “msName” and “msCategory” attributes to the instruments. (ms for morningstar)

Usage

```r
update_instruments.morningstar(Symbol, silent = FALSE)

update_instruments.ms(Symbol, silent = FALSE)
```
Arguments

Symbols character vector of Symbols of ETFs
silent silence warnings?

Value
called for side-effect.

Author(s)
Garrett See

References
http://www.morningstar.com

See Also
update_instruments.yahoo, update_instruments.TTR update_instruments.iShares

Examples
## Not run:
## backup .instrument environment
ibak <- as.list(FinancialInstrument:::instrument)
rm_instruments()
stock(s <- c("SPY", "USO", "LQD"), currency("USD"))
update_instruments.morningstar(s)
instrument.table(s)
## cleanup and restore instrument environment
rm_instruments(keep.currencies=FALSE)
loadInstruments(ibak)
## End(Not run)

update_instruments.yahoo

updates instrument metadata with data from yahoo

Description
Adds/updates information in instrument with data downloaded from yahoo

Usage

update_instruments.yahoo(Symbols = c("stocks", "all"), verbose = FALSE)

update_instruments.TTR(Symbols = c("stocks", "all"), exchange = c("AMEX", "NASDAQ", "NYSE"), silent = FALSE)
Arguments

Symbols can be a vector of instrument names, or, can be ‘all’ or ‘stocks’ or, for update_instruments.TTR, can be NULL in which case all stocks found with stockSymbols will be defined

verbose be verbose?

exchange character vector of names of exchanges. Used in ‘TTR’ method. Can be “AMEX”, “NASDAQ”, or “NYSE”

silent silence warnings?

Details

Although these functions are intended to update the metadata of previously defined instruments, update_instruments.TTR will define the stocks if they do not already exist.

update_instruments.TTR is only to be used on U.S. stocks denominated in USD.

Value called for side-effect

Author(s)

Garrett See

References


See Also

update_instruments.instrument, update_instruments.morningstar, update_instruments.masterDATA, stockSymbols, stock

Examples

## Not run:
stock('GS', currency('USD'))
update_instruments.yahoo('GS')
getInstrument('GS')
update_instruments.TTR('GS')
getInstrument('GS')

## End(Not run)
volep  

generate endpoints for volume bars

Description

generate endpoints for volume bars

Usage

volep(x, units)

Arguments

x  
time series containing 'Volume' column

units  
volume sum to mark for bars

Author(s)

Joshua Ulrich
Index

+Topic data
  currencies, 18
  root_contracts, 59
+Topic package
  FinancialInstrument-package, 3
    .get_rate, 7, 12
    .parseISO8601, 13
    .to_daily, 8
  add.defined.by, 9
  add.identifier, 9, 10, 36
  all.equal.instrument, 17, 18, 68
  alltick2sec (to_secBATV), 66
  assign, 32
  bond, 34
  bond (instrument), 31
  bond_series (future_series), 27
  build_series_symbols, 14, 16, 51
  build_spread_symbols, 15, 51
  buildBasket (buildSpread), 13
  buildHierarchy, 11, 20, 21, 35
  buildRatio, 7, 12, 24, 58
  buildSpread, 12, 13, 24, 25
  butterfly (synthetic), 64
  C2M, 16, 51
  CompareInstrumentFiles, 17
  currencies, 18
  currency, 32, 33, 36, 37, 56
  currency (instrument), 31
  Denotalize (Notionalize), 53
  exchange_rate, 18, 33, 56
  expires, 19
  expires.character, 20
  expires.instrument, 20
  FinancialInstrument
    (FinancialInstrument-package), 3
    find_instrument, 21
    fn_SpreadBuilder, 12–14, 23, 25
    format_id, 26, 50, 53, 57
    formatSpreadPrice, 24, 25
    fund (instrument), 31
    future, 29, 34, 51, 65
    future (instrument), 31
    future_series, 26, 27, 33, 34, 56, 63
    get, 13
    getInstrument, 4, 20, 22, 28, 32
    getOptionChain, 55
    getSymbols, 4, 13, 30, 39, 62
    getSymbols.FI, 29, 39, 61, 62
    grep, 21
    guaranteed_spread (synthetic), 64
    ICS (synthetic), 64
    ICS_root (synthetic), 64
    indexTZ, 30
    instrument, 9, 10, 20, 21, 30, 31, 33, 35, 38, 39, 55
    instrument.auto, 33
    instrument.table, 11, 21, 34
    instrument_attr, 4, 9, 10, 36, 62
    is.currency, 37
    is.currency.name, 37
    is.instrument, 38
    is.instrument.name, 38
    load_instruments, 4, 15, 16, 18, 33, 39, 59, 62
    loadInstruments, 30, 39, 62
    loadInstruments (saveInstruments), 59
    ls_AUD (ls_by_currency), 40
    ls_bonds (ls_instruments), 44
    ls_by_currency, 40
    ls_by_expiry, 42
ls_CAD(ls_by_currency), 40
ls_calls, 42
ls_calls(ls_instruments), 44
ls_CHF(ls_by_currency), 40
ls_currencies(ls_instruments), 44
ls_derivatives, 42
ls_derivatives(ls_instruments), 44
ls_EUR(ls_by_currency), 40
ls_exchange_rates(ls_instruments), 44
ls_expires(ls_expires), 43
ls_expires, 43
ls_funds(ls_instruments), 44
ls_future_series(ls_instruments), 44
ls_futures, 42
ls_futures(ls_instruments), 44
ls_FX(ls_instruments), 44
ls_GBP(ls_by_currency), 40
ls_guaranteed_spreads(ls_instruments), 44
ls_HKD(ls_by_currency), 40
lsICS(ls_instruments), 44
lsICS_roots(ls_instruments), 44
ls_instruments, 42, 44
ls_instruments_by, 47
ls_JPY(ls_by_currency), 40
ls_non_currencies(ls_instruments), 44
ls_non_derivatives(ls_instruments), 44
ls_NZD(ls_by_currency), 40
ls_option_series(ls_instruments), 44
ls_options, 42
ls_options(ls_instruments), 44
ls_puts, 42
ls_puts(ls_instruments), 44
ls_SEK(ls_by_currency), 40
ls_spreads(ls_instruments), 44
ls_stocks(ls_instruments), 44
ls_strikes, 48
ls_synthetics(ls_instruments), 44
ls_underlyings, 49
ls_USD(ls_by_currency), 40

M2C, 26, 51
M2C (C2M), 16
make.names, 30, 32
make_spread_id, 24, 50, 65
MC2N, 17
MC2N(month_cycle2numeric), 51
month_cycle2numeric, 26, 51, 52, 53
next_future_id, 51, 52
Notionalize, 53
option, 29, 34, 55, 65
option (instrument), 31
option_series, 26, 33, 34, 55, 56
option_series (future_series), 27
option_series.yahoo, 54
parse_id, 26, 33, 52, 55, 57, 63
parse_suffix, 26, 56, 56
prev_future_id (next_future_id), 52
quantmod, 4
redenominate, 7, 12, 57
regex, 21
reloadInstruments (saveInstruments), 59
rmbonds (ls_instruments), 44
rmb_by_currency (ls_by_currency), 40
rmb_by_expiry (ls_by_expiry), 42
rmb_currencies (ls_instruments), 44
rmb_derivatives (ls_instruments), 44
rmb_exchange_rates (ls_instruments), 44
rmb_funds (ls_instruments), 44
rmb_future_series (ls_instruments), 44
rmb_futures (ls_instruments), 44
rmb_FX (ls_instruments), 44
rmb_instruments (ls_instruments), 44
rmb_non_derivatives (ls_instruments), 44
rmb_option_series (ls_instruments), 44
rmb_options (ls_instruments), 44
rmb_spreads (ls_instruments), 44
rmb_stocks (ls_instruments), 44
rmb_synthetics (ls_instruments), 44
root_contracts, 59
save, 59
saveInstruments, 4, 17, 18, 59
saveSymbols.common (saveSymbols.days), 60
saveSymbols.days, 30, 60
setSymbolLookup, 33, 34, 62
setSymbolLookup.FI, 4, 30, 39, 62
sort_ids, 20, 63
source, 60
spread, 14, 33, 50, 51
spread (synthetic), 64
stock, 34, 73
INDEX

stock (instrument), 31
stockSymbols, 73
strptime, 30
strsplit, 65
synthetic, 64
synthetic.instrument, 24
Sys.Date, 15

to_secBATV, 66

update_instruments.instrument, 4, 67, 71, 73
update_instruments.iShares, 4, 69, 72
update_instruments.masterDATA, 69, 70, 73
update_instruments.md
   (update_instruments.masterDATA), 70
update_instruments.morningstar, 4, 69, 71, 73
update_instruments.ms
   (update_instruments.morningstar), 71
update_instruments.SPDR
   (update_instruments.iShares), 69
update_instruments.TTR, 4, 72
update_instruments.TTR
   (update_instruments.yahoo), 72
update_instruments.yahoo, 4, 68, 71, 72, 72

volep, 74

xts, 4