TODO

R topics documented:

When in br.desc 'CIE_DES_BULK' is not availbale use NA done set it in log allow br.desc/br.bulk.desc to use simulated mode Fix XXXX paragraphs

BloomR main functions

br.hist* function options

Historical data options

Description

Key/Values for `option.names`, `option.values` used by `br.hist*` function family.

Details

`option.names` and `option.values` are options vectors affecting the returned data. Options are set pairwise, for example to set `opt1` and `opt2` respectively to `val1` and `val2`, you would pass the arguments:

```
option.names=c("opt1", "opt2"), option.values=c("val1", "val2")
```

Here is a list of options:

periodicityAdjustment Determine the frequency and calendar type of the output. To be used in conjunction with `periodicitySelection`. If `ACTUAL`, it reverts to the actual date from today (if the end date is left blank) or from the End Date. If `CALENDAR`, (for pricing fields), it reverts to the last business day of the specified calendar period. Calendar Quarterly (CQ), Calendar Semi-Annually (CS), or Calendar Yearly (CY). If `FISCAL`, it reverts to the fiscal period end for the company - Fiscal Quarterly (FQ), Fiscal Semi-Annually (FS) and Fiscal Yearly (FY) only.

periodicitySelection Determine the frequency of the output. To be used in conjunction with periodicityAdjustment. if `DAILY`, `WEEKLY`, `MONTHLY`, `QUARTERLY`, `SEMI_ANNUALLY`, `YEARLY`.

currency Amends the value from local to desired currency. The value is a 3 letter ISO code string, e.g. USD, GBP. View `WCV<GO>` on the Bloomberg terminal for the full list.

overrideOption Indicates whether to use the average or the closing price in quote calculation. Values can be `OVERRIDE_OPTION_CLOSE` for using the closing price or `OVERRIDE_OPTION_GPA` for the average price.

pricingOption Sets quote to Price or Yield for a debt instrument whose default value is quoted in yield (depending on pricing source). `PRICING_OPTION_PRICE` sets quote to price; `PRICING_OPTION_YIELD` sets quote to yield.

nonTradingDayFillOption Sets to include/exclude non trading days where no data was generated. `NON_TRADING_WEEKDAYS` includes all weekdays (Monday to Friday); `ALL_CALENDAR_DAYS` includes all days of the calendar; `ACTIVE_DAYS_ONLY` includes only the days where the instrument and field pair were updated.

nonTradingDayFillMethod If data is to be displayed for non trading days what is the data to be returned. `PREVIOUS_VALUE` searches back and retrieve the previous value available for this security field pair. The search back period is up to one month. `NIL_VALUE` returns blank for the "value" value within the data element for this field.

maxDataPoints the maximum number of data points to return. If the original data set is larger, the response will be a subset, containing only the last `maxDataPoints` data points.

returnEids returns the entitlement identifiers associated with security. If TRUE, populates data with an extra element containing a name and value for the EID date.

returnRelativeDate returns data with a relative date. If TRUE, populates data with an extra element containing a name and value for the relative date. For example RELATIVE_DATE = 2002 Q2

adjustmentNormal Adjust for "change on day". If TRUE, adjusts historical pricing to reflect: Regular Cash, Interim, 1st Interim, 2nd Interim, 3rd Interim, 4th Interim, 5th Interim, Income, Estimated, Partnership Distribution, Final, Interest on Capital, Distribution, Prorated.

adjustmentAbnormal Adjusts for Anormal Cash Dividends. If TRUE, adjusts historical pricing to reflect: Special Cash, Liquidation, Capital Gains, Long-Term Capital Gains, Short-Term Capital Gains, Memorial, Return of Capital, Rights Redemption, Miscellaneous, Return Premium, Preferred Rights Redemption, Proceeds/Rights, Proceeds/Shares, Proceeds/Warrants.

adjustmentSplit Capital Changes Defaults. If TRUE, adjusts historical pricing and/or volume to reflect: Spin-Offs, Stock Splits/Consolidations, Stock Dividend/Bonus, Rights Offerings/Entitlement.

adjustmentFollowDPDF If TRUE (defaults) Follow the Bloomberg function as from `DPDF<GO>`.

CalendarCodeOverride Returns the data based on the calendar of the specified country, exchange, or religion. Value is a two character calendar code as from `CDR<GO>`. This will cause the data to be aligned according to the calendar and including calendar holidays. Only applies only to DAILY requests.

calendarOverridesInfo (Experimental, not tested) Returns data based on the calendar code of multiple countries, exchanges, or religious calendars as from `CDR<GO>`. This will cause the data to be aligned according to the set calendar(s) including their calendar holidays and only applies to DAILY requests. Requires `calendarOverrides`, which is a character vector of two-character calendar codes as from `CDR<GO>`; `calendareOverridesOperation`, which can be `CDR_AND` returning the intersection of trading days among multiple calendars or `CDR_OR` returning the union of trading days. That is, a data point is returned if a date is a valid trading day for any of the calendar codes specified in the request.

Overrides (Experimental, not tested) Append overrides to modify the calculation. `fieldID` specifies a field mnemonic or alpha-numeric, such as `PR092` or `PRICING_SOURCE`. Review FLDS for list of possible overrides. `value` sets the desired override value

Value

A data frame with historical data. If tickers are displayed, the first column shows tickers, the second one the time series dates and the following ones the values of the queried fields; otherwise the columns start with dates. Dates will also be shown as rows if `dates.as.row.names=TRUE`. If multiple tickers are queried, they are vertically stacked respecting the order in `securities` vector.

br.hist.csv

Historical data from grouped tickers in a CSV file
Reads a CSV file containing a group of tickers in each column and returns the historical data in xts or list format. The CSV file is assumed to have headers denoting group labels. It replaces 'br.bulk.csv"

Usage

```
br.hist.csv(con, file, field="PX_LAST", start=Sys.Date()-5, end.date=Sys.Date(),

                  cols=NULL, comma=TRUE,
                  addtype=FALSE, showtype=FALSE,
                  use.xts=TRUE, merge.xts=TRUE,

                  option.names = NULL, option.values = NULL,
                  only.trading.days = TRUE,

                  price=TRUE,
                  mean=ifelse(price, 10, 0.1), sd=1, jitter=0,
                  same.dates=FALSE, empty.sec=0,
                  weekend=TRUE, holidays=NULL)
```

Arguments

con the connection token returned from br.open(). If `NULL` simulated values are generated.
file path to CSV file.
field case insensitive string denoting the Bloomberg field queried. Defaults to "PX_LAST". If the field is wrong or not accessible, data will be empty, but no error will be raised.
start start date. Can be a Date object or an ISO string without separators (YYYYMMDD). Defaults to 5 days before current date.
end.date end date. Same format as `start`. Defaults to current date.
cols Logical or integer vector for selecting CSV columns (ticker groups). Defaults to all columns.
comma to be set to FALSE for (non-English) CSV, using semicolon as separator.
addtype If a string, it denotes the security type and is added to all tickers; if TRUE "Equity", will be added; if FALSE (the default), nothing will be added.
showtype if TRUE, security types will be removed from names of list or xts output. It defaults to FALSE.
use.xts if TRUE (the default) time series are formatted as xts objects. else as a data frame.
merge.xts if TRUE (the default) xts objects in the same group are merged using all rows and using NAs for missing observations.
option.names list of Bloomberg options names. Require `option.values` too.
option.values list of Bloomberg options values related to `option.names`.
only.trading.days if TRUE (the default) only trading days are used, else non-trading days are added as NA values.
price, mean, sd, jitter, same.dates, empty.sec, weekend, holidays arguments passed to `br.sample()` if `con=NULL`.

Details

Empty CSV cells or cells interpreted as NAs will be ignored.
If `con=NULL`, values are simulated by means of `br.sample()`. This function is used with default values, except for `start, end.date, price, mean, sd, jitter, same.dates, empty.sec, weekend, holidays`,

which can be explicitly passed as arguments, and `sec.names` depending on tickers found in the CSV file. These arguments are ignored if `con!=NULL`. See `br.sample()` help for more.

Value

a list where each element is the historical data of a CSV group.
If `use.xts=TRUE` and `merge.xts=FAlSE`, each group is a sub-list, whose elements are the the securoty time series as an xts object. If `use.xts=TRUE` and `merge.xts=TRUE`, each group is the merged xts object, obtained merging historical data of all securities of that group. If `use.xts=FALSE`, each group is a sub-list, where each element is the historical data of a security as a data frame.
If there is only one group, the first (and unique) element of the list will be returned (XXXXto check).

Demonstration

A sample CSV with Bloomberg tickers will look like follows:

```
read.csv("mybloomr/tickers.csv")
## This file is part of BloomR and anyway available here:
## https://github.com/AntonioFasano/BloomR/blob/master/res/tickers.csv
```

```
##          Financial     Technology      Indices
## 1   3988 HK Equity QCOM US Equity    DJI Index
## 2      C US Equity CSCO US Equity DJUSFN Index
## 3 601288 CH Equity  700 HK Equity  W1TEC Index
## 4    BAC US Equity  IBM US Equity
## 5   HSBA LN Equity INTC US Equity
```

Note:

- CSV group headers are mandatory;
- Group headers need not to be the same length.

We can now download data:

```
br.simulate(is=TRUE) # Simulated mode: replace TRUE with FALSE on terminal
```

```
data=br.hist.csv(con, "mybloomr/tickers.csv")
```

```
## Processing Financial ...
## Loading 3988 HK Equity
## Loading C US Equity
## Loading 601288 CH Equity
## Loading BAC US Equity
## Loading HSBA LN Equity
## Processing Technology ...
## Loading QCOM US Equity
## Loading CSCO US Equity
## Loading 700 HK Equity
## Loading IBM US Equity
## Loading INTC US Equity
```

```
## Processing Indices ...
## Loading DJI Index
## Loading DJUSFN Index
## Loading W1TEC Index
```

Above you see some info about data being processed that we will not show anymore in the following.

If you want to have detailed ticker descriptions, see br.bulk.desc Example. Downloaded data look like follows:

```
data
```

```
## $Financial
##            3988 HK   C US 601288 CH BAC US HSBA LN
## 2017-06-21      NA     NA        NA     NA   9.856
## 2017-06-22      NA 10.328        NA     NA  10.796
## 2017-06-23      NA     NA    10.348     NA   9.192
##
## $Technology
##            QCOM US CSCO US 700 HK IBM US INTC US
## 2017-06-21      NA  11.532     NA  8.804      NA
## 2017-06-22  10.745      NA     NA     NA      NA
## 2017-06-23      NA      NA  9.851     NA  10.392
##
## $Indices
##              DJI DJUSFN  W1TEC
## 2017-06-21    NA  9.567 12.000
## 2017-06-22    NA 11.391     NA
## 2017-06-23 10.79     NA  8.729
```

Note:

- The name of the securities tickers is stored without the security type: “Equity”, “Index”, etc. If this piece of info is significant for you, pass `showtype = TRUE`.
- Time series start date defaults to 5 days before current date, unless you set `start` to: an R Date object (`start=as.Date("2014/9/30")`) or to a more friendly ISO string (`start="20140930"`).

Data are stored as a list of xts objects, each representing one group of tickers in the CSV file.

```
length(data)
```

```
## [1] 3
```

```
names(data)
```

```
## [1] "Financial"  "Technology" "Indices"
```

```
class(data$Financial)
```

```
## [1] "xts" "zoo"
```

If you prefer you may get time series as data frames, and precisely as a list representing the ticker groups, where each group is in turn a list containing a data frame for each security:

```
data=br.hist.csv(con, "mybloomr/tickers.csv", use.xts=FALSE)
```

```
length(data)
```

```
## [1] 3
```

```
names(data)
```

```
## [1] "Financial"  "Technology" "Indices"
```

```
class(data$Financial)
```

```
## [1] "list"
```

```
length(data$Financial)
```

```
## [1] 5
```

```
names(data$Financial)
```

```
## [1] "3988 HK"   "C US"      "601288 CH" "BAC US"    "HSBA LN"
```

```
class(data$Financial$`BAC US`)
```

```
## [1] "matrix"
```

By defaults time series list values from the Bloomberg "PX_LAST" field. To change the default field use:

```
data=br.hist.csv(con, "mybloomr/tickers.csv", field = "PX_OPEN")
```

You can choose to import only some of the CSV groups

```
## Processing Financial ...
## Loading 3988 HK Equity
## Loading C US Equity
## Loading 601288 CH Equity
## Loading BAC US Equity
## Loading HSBA LN Equity
## Processing Indices ...
## Loading DJI Index
## Loading DJUSFN Index
## Loading W1TEC Index
```

```
data=br.hist.csv(con, "mybloomr/tickers.csv", cols=c(1,3))
## or equivalently:
data=br.hist.csv(con, "mybloomr/tickers.csv", cols=c(TRUE, FALSE, TRUE))
```

```
names(data)
```

```
## [1] "Financial" "Indices"
```

In the CSV file, if your tickers represent all equities, you can omit the type.

Consider this CSV:

```
read.csv("mybloomr/tickers.eqt.csv")
## This file is part of BloomR and anyway available here:
## https://github.com/AntonioFasano/BloomR/blob/master/res/tickers.eqt.csv
```

```
##   Financial Technology
## 1   3988 HK    QCOM US
## 2      C US    CSCO US
## 3 601288 CH     700 HK
```

```
## 4    BAC US     IBM US
## 5   HSBA LN    INTC US
```

Note how the "Equity" type is missing! But you can use this CSV file with `addtype`:

```
data=br.hist.csv(con, "mybloomr/tickers.eqt.csv", addtype=TRUE)
```

Before *going home*, don't forget to:

```
br.close(con)
```

```
## Error in br.close(con): object 'con' not found
```

br.bulk.desc

Description

Get security descriptions for a vector of tickers.

Usage

```
br.bulk.desc(con, tiks)
```

Arguments

con the connection token returned from br.open()
tiks character vector of the tickers queried for data

Value

A list of data frames, each representing the description of a security. For the format of data frames see the function `br.desc`.

Example

```
con=br.open()
data=read.csv("mybloomr/tickers.csv", as.is=TRUE)
br.bulk.desc(con, as.vector(as.matrix(data[1:2,])))
br.close(con)
```

br.idx

Description

Returns the historical data for the constituents of an index in xts or list format. It replaces `br.bulk.idx`.

Usage

```
br.idx(con, index, field="PX_LAST", start=Sys.Date()-7, end.date=Sys.Date(),

            include.idx=TRUE, showtype=FALSE,
            use.xts=TRUE, merge.xts=TRUE,

            option.names = NULL, option.values = NULL,
            only.trading.days = TRUE,

            nsec=10, sec.names = NULL,

            price=TRUE,
            mean=ifelse(price, 10, 0.1), sd=1, jitter=0,
            same.dates=FALSE, empty.sec=0,
            weekend=TRUE, holidays=NULL)
```

Arguments

con the connection token returned from br.open(). If `NULL` simulated values are generated.
index string denoting the index ticker with or without the final security type label ('Index')
include.idx if TRUE (default) returns also historical data for the index.
nsec number of simulated index constituents. Ignored if `con!=NULL`, it defaults to 10.
sec.names character vector with names of sampled index constituents. Ignored if `con!=NULL`. By default security names are like 'memb1', 'memb2', etc.

For other arguments see the function `br.hist`.

Details

If `con=NULL`, values are simulated by means of `br.sample()`. This function is used with default values, except for `nrow, nsec1, price, start, same.dates, no.na, empty.sec, sec.names`.

Value

If `use.xts=FALSE`, a list where each element is the historical data of a constituent as a data frame.
If `use.xts=TRUE` and `merge.xts=FAlSE`, a list where each element is the historical data of a constituent as an xts object.
If `use.xts=TRUE` and `merge.xts=TRUE`, an xts oject where where each column is the historical data of a constituent .
If `include.idx=TRUE`, the last column or element will be the historical data of the index.

br.hist

Historical data for vector of tickers
Returns the historical data for a vector of tickers in xts or list format. It replaces 'br.bulk.tiks"

Usage

```
br.hist(con, tiks, field="PX_LAST", start=Sys.Date()-7, end.date=Sys.Date(),

              addtype=FALSE, showtype=FALSE,
              use.xts=TRUE, merge.xts=TRUE,

              option.names = NULL, option.values = NULL,
              only.trading.days = TRUE,

              price=TRUE,
              mean=ifelse(price, 10, 0.1), sd=1, jitter=0,
              same.dates=FALSE, empty.sec=0,
              weekend=TRUE, holidays=NULL)
```

Arguments

tiks Character vector of the tickers queried for data
start Start date can be a POSIXlt/ct, Date, ISO string, UK string with slashes or dashes
start same format of start
use.xts if TRUE (the default) time series are formatted as xts objects else as a data frame.
merge.xts if TRUE (the default) xts objects are merged using all rows and using NAs for missing observations.

For other arguments see the function `br.hist.csv`.

Details

If an element of `tiks` is `NA` or empty (`""`) it is ignored. This is intended to avoid errors when the character vector are read from a CSV file with empty cells.
If `con=NULL`, values are simulated by means of `br.sample()`. Sampled values are based on default values of `br.sample()`, but it is possible to set explicitly `start, end.date, price, mean, sd, jitter, same.dates, empty.sec, weekend, holidays`; `sec.names` depends on `tiks` argument. These arguments are ignored if `con!=NULL`. See `br.sample()` help for more.

Value

If `use.xts=FALSE`, a list of character matrices, where the first column, named "date", has the observation dates, the second column, named after the field, has field values. The list names are the tickers. Empty time series are returned as NULL. If all time series are empty a list of NULLs is returned.

If `use.xts=TRUE` and `merge.xts=FAlSE`, a list of xts objects, where the xts index has the observation dates and its data column, named after the field, has field values. The list names are the tickers. Empty time series are returned as NA. If all time series are empty a list of NAs is returned.

If `use.xts=TRUE` and `merge.xts=TRUE`, then when:
A) There is at least one non-empty TS, an xts object is returned, where the index has the observation dates and columns, named after the tickers, have field values. Empty time series are returned as a NA column for the related xts ticker. B) All time series are empty a vectors of NAs of the same length as the queries tickers is returned.

Example

```
con=NULL # Open simulated connection and  load some data
br.hist(con, c("MSFT US", "AMZN US"), addtype=TRUE)
```

```
## Loading MSFT US Equity
```

```
## Loading AMZN US Equity
```

```
##            MSFT US AMZN US
## 2017-06-19      NA  10.108
## 2017-06-20      NA  10.003
## 2017-06-21   8.639  10.394
## 2017-06-22   9.009  10.536
```

```
br.close(con) # Use the token to release the connection
```

See Also

br.hist.csv

br.desc

Description

Get security descriptions.

Usage

```
br.desc(con, tik)
```

Arguments

con the connection token returned from br.open()
tik string denoting the ticker queried for data

Value

A data frame containing the value of the Bloomberg fields form `ds001` to `ds009` and the long field `CIE_DES_BULK`.

br.md2pdf

Description

Make a markdown file into a PDF It assumes that you have installed the BloomR LaTeX addons

Usage

```
br.md2pdf(md.file, pdf.file)
```

Arguments

md.file path to the markdown file to be converted.
pdf.file path to the PDF file to be generated.

Details

The function will stop with an error if you have not installed BloomR LaTeX addons. To install them use `br.getLatexAddons()`.

Value

If there are no errors, it returns zero invisibly, otherwise it prints an error message and returns the related error code.

br.sample

Description

Return simulated historical data for n securities in xts or df format.

Usage

```
br.sample(nrow=NULL,  price=TRUE,
               start=Sys.Date() - 7, end.date=Sys.Date(),
               field="FIELD",
               use.xts=TRUE,
               mean=ifelse(price, 10, 0.1), sd=1, jitter=0,
               rand.dates=TRUE, weekend=TRUE, holidays=NULL)
```

Arguments

nrow number of simulated data points for each security; if `same.dates=FALSE`, the number of rows for each sampled security will be a random number not exceeding nrow, else it will be nrow for all securities. Actual number of rows depends on the value of `rand.dates`, `weekend`, `holidays`.
price if TRUE (default), simulated values are non-negative.
start start date. Can be a Date object or an ISO string without separators (YYYYMMDD). Defaults to current date.
end.date end date. Same format as `start`. Defaults to current date.
field case insensitive string denoting the Bloomberg field queried. Defaults to "FIELD".
use.xts if TRUE (the default) time series are formatted as xts objects else as a data frame.
mean mean of security generated values. If `price=TRUE`, default to 10 else defaults to 0.1.
sd sd of security generated values. It defaults to 1.

jitter modifies each security mean by adding adding a random value in [-jitter, jitter]. Defaults to 0.

rand.dates if TRUE, all sampled securities will refer to the same dates and for each security the number will equal nrow. If FALSE (default), date values and number will randomly differ. For each security the random number will not exceed `nrow`.

weekend if TRUE (default), weekend dates are removed.

holidays list of dates to be removed,

Details

`br.sample()` assumes by default that data for some securities might not be available on certain days and time series might be misaligned (see "Missing observations and misalignment" in `br.hist()`), therefore the date values and count for each time series generated will randomly differ, with `nrow` as the maximum number of days. If you want all time series to share tha same dates, set `rand.dates=FALSE`. In this case, time series produced are aligned and you don't see any merge NA, the acutal dates generated depends on the value of `weekend` and `holidays`. If there are no holidays falling in time windows queried and `weekend=FALSE` the number of generated dates equals `nrow`.

Value

If `use.xts=FALSE`, a data frame object, where the first column is the vector with all generated dates merged and each subsequent column contains the sampled data of a security. If `use.xts=TRUE`, an xts object, where each element is the sampled data of a security, while the dates will be part of the xts time object. In both cases if `rand.dates=TRUE` generated data points might likely have different length

XXXX and the the date gaps will be filled with NAs, except if `no.na=TRUE`. If the generated values are only NAs the output will be converted to a 0-rows xts or data frame, containing only security labels accessible with `dimnames(*)[[2]]`.

Internal BloomR functions

Usage

Description:

Internal functions not to be used by the end user

Usage:

```
.br.is.con(con)
.br.types
.br.check.type(type)
.br.cuttype(type)
.br.jar()
.br.test.dates(start, end, holidays=NULL, asChar=FALSE)

.br.raw(security, field="PX_LAST",
    ## Start/End date can be a POSIXlt/ct, Date, ISO string, UK string with slashes or dashes.
    start.date, end.date=Sys.Date(),
    alldays=FALSE, optnams=NULL, optvals=NULL, ovrnams=NULL, ovrvals=NULL)
```

```
.br.choose.testdata(security, field="PX_LAST",
    startDate, endDate,    # ISO string
    alldays=FALSE, optnams=NULL, optvals=NULL, ovrnams=NULL, ovrvals=NULL)
.br.developer(mode=TRUE)
.br.is.dev()

.br.wpath(path) TODO XXX
```

Arguments:

con the connection token returned from br.open()
type a string representing the security type
mode TRUE if in developer mode
alldays equivalent to `!only.trading.days` in `br.hist`
optnams, optvals, ovrnams, ovrvals equivalent to `option.names`, `option.values`, `override.names`, `override.values` in `br.hist`

See `br.hist`, for other `.br.raw` arguemnts

Details

`.br.is.con` checks for the validity of a connection token. `.br.types` is a character vector with security types suitable as an argument for BloomR multi-ticker functions. `.br.check.type` checks if a type matches `.br.types`. `.br.cuttype` cuts trailing security type from character vector. `.br.jar()` returns the path to the blpapi*.jar `.br.session` object storing BloomR session information. `.br.test.dates` tests the validity of given start, end date, possibly against a holiday vector.

`.br.developer` sets developer which modifies the behaviour of some functions for debugging. `.br.is.dev` tests if in developer mode

`.br.raw` gets low level historical data. It might use stored local data in developer mode or call `.br.raw_` for actual low level download. `.br.choose.testdata` chooses what test data to use, when in developer mode, based on arguments or `TNAMS` character vector if found in the global environment.

Manage connections

Description

Open, close and test the connection to the Bloomberg service.

Usage

```
br.open()
br.close(con)
br.simulate(is=TRUE)
br.is.sim()
```

Arguments

con the connection token returned from br.open()

is if TRUE (default), simulate connection.

Details

`br.open` returns the connection token needed by the BloomR function downloading data. When you finish you session, you pass it to `br.close`. If you have run `br.simulate(is=TRUE)`, data are simulated (and your connection token is `NULL`). `br.is.sim()` tests if the connection is simulated.
If `br.is.sim()`, closing the connection is optional. Anyway running `br.close(con)`, even if `con==NULL` avoids adding this line when you switch to a actual data download.

Example

```
con=br.open() # Open the connection and get the token and load some data
br.hist(con, c("MSFT US", "AMZN US"), addtype=TRUE)
br.close(con) # Use the token to release the connection
```

Misc functions

Description

`rm.all` deletes all objects (variables and functions) from memory, including invisible objects (those starting with a dot). `rm.var` deletes non-function objects from memory.

Usage

```
rm.all()
rm.var()
```

Beta functionalities

Description

Activate beta functionalities, if available for this release.

Usage

```
br.beta()
```

Beta misc functions

Description

Miscellaneous functions dealing with dates.

Usage

```
br.try.date(d)
br.is.same.class(...)
```

Arguments

d a POSIXlt, POSIXct, Date, "%Y/%m/%d", or "%Y-%m-%d" vector

Details

`br.try.date` converts a vector to a date vector if possible or return `NULL`. Any vector element should be POSIXlt, POSIXct, Date, "%Y/%m/%d", or "%Y-%m-%d"

`br.is.same.class` check if all supplied argumets have the same class. It is mostly intended to check if dates are homogeneous.

Deprecated functions

Description

Functions (planned to be) deprecated.

Usage

```
.br.sample.deprecated(nsec=NULL, no.na=NULL, df=NULL,
                 sec.names=NULL, empty.sec=NULL, same.dates=NULL)
```

Arguments

See current non beta BloomR.

Time extension functions

Description

Functions to get, set dates.

Usage

```
day(d)
month(d)
year(d)
day(d, n)
month(d, n)
year(d, n)
```

```
day(d)=x
month(d)=x
year(d)=x
d %+% n
d %-% n
last.day(d)
day.us(d1, d2)
```

Arguments

d, d1, d2 objects of class date
x an integer representing the day/month/year
n an integer representing the months to add/subtract

Details

If `component` is `day`, `month` or `year`: `component(d)` returns the *component* of the date `d` as an integer; `component(d, n)` returns the date `d` with the *component* set to the integer `n`; `component(d)= n` sets to the *component* of the date `d` to the integer `n`.
`%+%` and `%-%` add and subtract months to a date.
`last.day` returns last day of the month as an integer. `day.us` calculates date differences with the US convention.