BloomR facility functions

R topics documented:

br.bulk.csv

Historical data from grouped tickers in a CSV files
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.

Usage

```
br.bulk.csv(con, file, start = Sys.Date() - 5, field = "PX_LAST",
    cols = NULL, addtype = FALSE, showtype = FALSE, use.xts = TRUE,
    comma = TRUE,
    price=TRUE, nrow=5, same.dates=FALSE, no.na=FALSE, empty.sec=0
    )
```

Arguments

con the connection token returned from br.open(). If `NULL` simulated values are generated.

file
path to CSV file.

start
start date. Can be a Date object or an ISO string without separators. Defaults to 5 days before current date.

field
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.

cols
Logical or integer vector for selecting CSV columns (ticker groups). Defaults to all columns.

addtype
If a string denoting the security type, it will be 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) each group will be formatted as an xts object else as a list.
comma
to be set to FALSE for (non-English) CSV, using semicolon as separator.
nrow
maximum number of simulated rows (actual is random). Ignored if `con!=NULL`, it defaults to 5.
empty.sec
ratio of securities returning no data. Ignored if `con!=NULL`, it defaults to 0.

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 `nrow, start, same.dates, no.na, empty.sec`, 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`, elements are xts object, where each column is the historical data of a security.
If `use.xts=FALSE`, elements are sub-list, where each element is the historical data of a security.
If there is only one group, the first (and unique) element of the list will be returned.

Demonstration

A sample CSV with Bloomberg tickers will look like follows:

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

Note:

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

We can now download data:

```
con=NULL
```

```
data=br.bulk.csv(con, "tickers.csv")
```

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
## 2014-12-06  10.295    NA     8.100     NA      NA
## 2014-12-07   9.508    NA     9.517 10.096   9.499
## 2014-12-09  10.549    NA    10.654     NA      NA
## 2014-12-10   7.800 8.973        NA  9.472   9.077
##
## $Technology
##            QCOM US CSCO US 700 HK IBM US INTC US
## 2014-12-06  10.259   9.837 11.144  9.333      NA
## 2014-12-07   9.913      NA     NA 10.200      NA
## 2014-12-08   8.732      NA     NA 10.416      NA
## 2014-12-09      NA      NA 10.061 10.240      NA
## 2014-12-10      NA      NA     NA  8.636  12.101
##
## $Indices
##               DJI DJUSFN  W1TEC
## 2014-12-06  9.450     NA     NA
## 2014-12-07 10.124 12.361 10.687
## 2014-12-08     NA  9.827     NA
## 2014-12-09  8.969 10.352 10.958
## 2014-12-10  9.298     NA 12.392
```

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.bulk.csv(con, "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] "data.frame"
```

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

```
data=br.bulk.csv(con, "tickers.csv", field = "PX_OPEN")
```

You can choose to import only some of the CSV groups

```
data=br.bulk.csv(con, "tickers.csv", cols=c(1,3))
## or equivalently:
data=br.bulk.csv(con, "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("tickers.eqt.csv")
## This file is part of BloomR and anyway available here:
## https://github.com/AntonioFasano/BloomR/blob/master/res/tickers.eqt.csv
```

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

```
data=br.bulk.csv(con, "tickers.eqt.csv", addtype=TRUE)
```

Before going home, don't forget to:

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

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

br.bulk.idx

Description

Returns the historical data for the constituents of an index in xts or list format.

Usage

```
br.bulk.idx(con, index, start = Sys.Date() - 5, field = "PX_LAST",
     showtype = FALSE, include.idx = TRUE, use.xts = TRUE, nsec = 50,
     price = TRUE, nrow = 5, same.dates=FALSE, no.na=FALSE, empty.sec = 0,
     sec.names = 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.bulk.csv`

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=TRUE`, an xts object, where each column is the historical data of a constituent.
If `use.xts=FALSE`, a list, where each element 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.bulk.tiks

Bulk historical data
Returns the historical data for a vector of tickers in xts or list format

Usage

```
br.bulk.tiks(con, tiks, start=Sys.Date()-5, field="PX_LAST",
    addtype=FALSE, showtype=FALSE, use.xts=TRUE,
    price=TRUE, nrow=5, same.dates=FALSE, no.na=FALSE, empty.sec=0)
```

Arguments

tiks character vector of the tickers queried for data

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

Details

If an element of `tiks` is `NA` or empty (`""`) it is ignored. This is intended to avoid errors when the cahracter 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, same.dates, no.na, empty.sec`; `sec.names` depends on `tiks` argument. These arguments are ignored if `con!=NULL`. See `br.sample()` help for more.

Value

If `use.xts=TRUE`, an xts object, where each column is the historical data of a security.
If `use.xts=FALSE`, a list, where each element is the historical data of a security.

Example

See Also

br.bulk.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.sample

Description

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

Usage

```
br.sample(nrow, nsec=1, price=TRUE, start=Sys.Date(),
mean=ifelse(price, 10, 0.1), sd=1, jitter=0, same.dates=FALSE, no.na=FALSE,
empty.sec=0, df=FALSE, sec.names=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.

nsec
number of simulated securities (defaults to 1).

price
if TRUE (default), simulated values are non-negative.

start
start date. Can be a Date object or an ISO string without separators. Defaults to current date.

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.

same.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`.

no.na
if `same.dates=FALSE`, when merging sampled security data NAs are likely to be produced. If `no.na=FALSE` (default) they will be left, otherwise they will be removed using R `na.omit`

df
if FALSE (default), the output will be an xts object, else the output will be a data frame with the first column containing the dates of the sampled data.

sec.names
character vector for column names. If `df=FALSE` the length of the vector should be equal to `nsec`, else to `nsec + 1` (because of the first column containing dates). By default security names are like 'sample1', 'sample2', etc. and the date column is named 'date'.

empty.sec
ratio of securities returning no data (defaults to 0). The result is rounded without decimal places.

Value

If `df=TRUE`, 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 `df=FALSE`, 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 `same.dates=FALSE` and/or `empty.sec!=0` generated data points will have different length 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]]`.

Deprecated functions

Description

Functions not used anymore generating an informative error

Usage

```
bbg.open()
```

```
bbg.close(con)
```

Arguments

con the connection token returned from br.open()

Example

```
con=bbg.open()
##  Sorry 'bbg.open' is now deprecated. Please use br.open().
```

Internal BloomR functions

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()
```

Arguments:

con the connection token returned from br.open()
type
a string representing the security type

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 `br.bulk*` 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

Manage connections

Description

Open and close the connection to the Bloomberg service.

Usage

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

Arguments

con the connection token returned from br.open()

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 are using simulated data and so your connection token is NULL, 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

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()
```

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.