Package 'rtws'

January 21, 2025

Title R api to Interactive brokers TWS

Version 0.7.5.000

Description This package provides access to the Interactive Brokers (IB) TWS API directly from R. It uses the formal Python api maintained by IB to emulate asynchronuous threaded Python processes on the single-thread R platform.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 3.5.0)

Imports reticulate, timeDate, R6,

Suggests methods

RoxygenNote 7.2.3

LazyData true

R topics documented:

`.rtws-package` **A TWS(Trader WorkStation) API for R**

Description

The rtws package provides an interface for accessing Interactive Brokers' TWS API from R. Powered by the reticulate package, rtws runs IB's Python API in a multi-threaded Python environment and facilitates asynchronous communication with IB's servers.

Developed primarily for private use, the package implements only a subset of the functions and callbacks available in the official Python API. However, its structure has been designed to allow for easy extension.

Disclaimer

This library is provided "as is" without any guarantees, whether explicit or implied. The authors disclaim all liability or warranty regarding the software's information, technology, functionality, or stability. Under no circumstances shall the authors be held responsible for direct or indirect trading losses incurred by relying on this software or its accompanying documentation. Please thoroughly test the software in a simulated (paper) trading environment and always conduct your own analysis before making any investment decisions. None of the materials, including this document and related guides, constitute an offer, solicitation, advertisement, or investment advice. Any examples, strategies, or trading ideas presented are for illustrative and educational purposes only and should not be interpreted as indicators of past or future performance or the likelihood of success of any specific investment or strategy.

Details

Installation

- Step 1: Download and Install the Package

Download the package's tar.gz file and install it in R:
`> install.packages("rtws.nn.nn.nnn.tar.gz", "repos=NULL)`

- Step 2: Install reticulate

Ensure the reticulate package is installed in R. If not, install it using:
`> install.packages("reticulate")`

or, as root directly from package management
`$apt install r-cran-reticulate`

- Step 3: Install Python and IB Python API

Make sure Python 3 and pip3 are installed on your system. On most server platforms, these are pre-installed. If not, refer to the reticulate documentation for troubleshooting.

Next, download and install the official IB Python API from Interactive Brokers API. You can download it from <`https://interactivebrokers.github.io/`>. Follow the steps for installation.

You will also need to configure your TWS settings, such as port, trusted IP addresses, and API permissions. Detailed instructions are available at <https://www.interactivebrokers.com/campus/ibkr-api-page/twsapi-doc/#tws-settings

- Step 4: Initialize the System

To initialize the system, load the library and connect:

```
> library(rtws)
> install_rtws()
> conn(host = "x.x.x.x", port = 0000L, clientId = 19L, verbose = 1)
```

Here, host is the IP address of the machine running TWS, and port is the one configured in TWS settings. If successful, rtws will return:
`Connection ready with oid [int]`

Architecture

The library offers three main categories of functions:

Blocking API functions Blocking functions return requested data immediately. These are suitable for requests involving static information, where asynchronous operations offer little benefit. Examples include:

- reqIds
- searchSymbols
- accountUpdates
- contractDetails
- historicalData
- completedOrders

non-blocking API functions Non-blocking functions leverage asynchronous data exchanges, running the Python API in the background as a thread (similar to a daemon). Data from callbacks is buffered and can be accessed explicitly. Non-blocking functions include:

- headTimeStamp
- realTimeBars
- mktData
- placeOrder
- openOrders

To retrieve buffered data, use:

- read.nb

Remember that requests persist until explicitly canceled. To prevent overflow, always cancel non-blocking requests and clean up with read.nb after cancellation.

Object preparation functions These functions create Python objects required for API calls but do not directly interact with TWS.

- cntr
- ordr
- efltr
- pprint
- py2list

accountUpdates *request account status* — **blocking**

Description

Request curennt account and securities portfolio status

Usage

```
accountUpdates(acct)
```

Arguments

acct	character account name

Value

returns a list with a) detailed account status and b) the account's securities portfolio positions.

Examples

```
## Not run:
acc <- accountUpdates("<account number or alias>")

## End(Not run)
```

contractDetails *get contract details* - — **blocking**

Description

request contract details with sparse data request delivery of contract and associated information

This function can be used to look for all contacts that fulfill certain min criteria as suitable contracts. Operates on a list of contracts, too.

Usage

```
contractDetails(cnt)
```

Arguments

cnt	a contract object

Value

a nested list with contract attributes, contract details and the contract as Python object for contract.details result object. Contracts can be extracted using cntrd2cntr on this.

Examples

```
## Not run:
# one contract returned
c1  <- cntr(symbol="ES", localSymbol="ESH5", exchange = "CME", secType="FUT")
cd

# multiple contracts returned
c2  <- cntr(symbol="BAYN", secType="STK", exchange="SMART")
cd

## End(Not run)
```

---

head-timestamp *head timestamp request* — **non-blocking**

Description

Request/cancel headTimeStamp subscriptions

Usage

```
headTimeStamp(cnt, whatToShow = "TRADES", useRTH = 0L)

cancelHeadTimeStamp(reqids)
```

Arguments

cnt	list with a contract object
whatToShow	whatToShow, e.g. "TRADES", "MIDPOINT", "ASK", ...
useRTH	looks only within regular trading hours
reqids	request ids used for subscription

Details

IB has implemented the headTimeStamp request as subscription. For the moment it was kept this way, although it is difficult imagining a realistic use-case, which would benefit from this design choice. This function might be changed to blocking in some time, so please use with care.

Value

headTimeStamp returns the earliest observation timestamp for the contract under the specified conditions

`historicalData` *get historical market data - —* **blocking**

Description

request historic market data used to obtain historic market data for a security

historic data is returned as bar data (open, high, low, close) plus counter of how many transactions occured during the bar period. The bar witdh, duration are defined by the userwithin the limitations of the acount used at IB.

Usage

```
historicalData(
  cnt,
  endDateTime = "",
  durationStr = "10 D",
  barSizeSetting = "1 hour",
  whatToShow = "TRADES",
  useRTH = "0"
)
```

Arguments

`cnt`	a contract object
`endDateTime`	the time and date as (yyyymmdd hh:mm:ss) up to when historic data should be delivered. Defaults to "", meaning up to now.
`durationStr`	a string specifying the duration from first to last bar, e.g. "3 M" for three months
`barSizeSetting`	the with of a single bar, starting with one sec "1 sec"
`whatToShow`	which data to return, e.g. "TRADES","MIDPOINT","BID",...
`useRTH`	should data for regular trading hours only be retuned (True, default)

Details

Some comments on the arguments for this request: endDateTime expects a character formatted "YYYYmmdd HH:MM:SS". durationStr consists of a integer number, then then one of: S (Second(s), D (Day(s)), W (Week(s)), M (Month(s)), Y (Year(s)) for example "8 W" for eight weeks. barSizeSettings accepts the following values:

1, 5, 10, 15, 30	secs
1, 2, 3, 5, 10, 15, 20, 30	mins
1, 2, 3, 4, 8	hrs
1	days
1	weeks
1	months

Value

returns data frame with the time series

Examples

```
## Not run:

c1  <- cntr(symbol="ES", localSymbol="ESH5", exchange = "CME", secType="FUT")
hd  <- historicalData(c1)
hd

## End(Not run)
```

`install-and-connect` *setup rtws environment*

Description

`install_rtws' creates a python environment with API functions that can called from R. \cr connstarts thr` disconnects threads from TWS and stops them.

Usage

```
install_rtws(py_env = "rtws")

conn(host = NULL, port = NULL, clientId = NULL, verbose = NULL)

disconn()
```

Arguments

`py_env`	Python virtual env, defaults to "rtws". It is not recommmended to change the parameter.
`host`	server on which tws is running, default home
`port`	port number of tws host, default 7891L
`clientId`	Client Id for Tws, default 145L
`verbose`	verbose level

Details

`install_rtws` creates a python environment acessible from R with reticulate. Prepares the environment by importing necessary modules, especially the official IB API, and sources the rtws scripts providing the core functionality Needs to be called just once after the `rtwd` library has been loaded. `conn` starts a background API thread and connects the thread to TWS. `disconn` disconnects threads from TWS and stops them.

Value

called for their side-effects

mkt-tick-data *Request tick data and cancel tick-data subscriptions* — **non-blocking**

Description

mkdData requests a subscription to tick data, 'cancelMktData' cancels a subscription

Usage

```
mktData(cnt, genTickTypes = "", snapshot = F)

cancelMktData(reqids)
```

Arguments

cnt	a contract object
genTickTypes	optional list of one or more aditional generic ticks
snapshot	• only one-time current data for security (no cancelMktData required)
reqids	request ids to cancel

Details

mktData requests tick data for a single instrument. However, often more than one security needs to be monitored. This can be achieved by successively calling this function, see examples. cancelMktData accepts a list of reqids.

Value

dataframe with ticks and maching reqId

Examples

```
## Not run:
c1   = cntr(symbol="AAPL",exchange="SMART",secType="STK",currency="USD")
## snapshot
rq <- mktData(c1, "snapshot")
Sys.sleep(1)
snap <- read.nb(rq = rq)

## multiple securities
## Since the call is non-blocking, it returns immediately. Calling `read.nb`
## at any later point in time will return the accumulated ticks as `data.frame`.
## Note that reqids and contracts have the same ordering, thus each reqId
## references a one contract.The resulting data.frame therefore includes a column
## with the reqIds allowing to map the contract to the data.

c2   = cntr(symbol="EUR",currency="USD",secType="CASH",exchange="IDEALPRO")
reqids <- lapply(list(c1,c2), mktData)
```

```
Sys.sleep(5)
cancelMktData(reqids)
tcks <- read.nb(unlist(reqids))

## End(Not run)
```

`object-generation` *functions to handle python objects*

Description

`cntr` allows setting up a contrct object

- Any contract attrubute may be given as a named argument (see details). Please refer to IB TWS Api documntation for more information.
- Typically minimum requirements for a valid contract are symbol, the exchange and the secType. There are however no guarantees. `ordr` create an order object manually
- This function creates an order object from scratch. òrdr‘ accepts any order attributes, which should be given as named arguments.
- The set of possible attributes of the order object is huge and not listed here. A minimal set or attributes is presented in the details section. `efltr` defines an executionFilter object that is required for execReports requests.

Usage

```
cntr(...)

ordr(...)

efltr(...)
```

Arguments

`...`	named atributes of the object

Details

A Contract can defined by the following attributes that need to be named parameters. Often symbol or localSymbol, exchange and secType are sufficuent.

- character conId
- character symbol
- character secType
- character lastTradeDateOrContractMonth
- float strike
- character right

- character multiplier
- character exchange
- character primaryExchange
- character currency
- character localSymbol
- character tradingClass
- character includeExpired
- character secIdType
- character secId
- character description
- character issuerId

Alternatively, conractDetails can be user to get all matching contracts.

The IB order object has more than 150 attributes, which cannot be listed here. The read more about the API order object please refer to IB's documentation. However, here is the minimal set of attributes to submitt an order:

- self.orderId required
- self.clientId optional
- self.action BUY or SELL
- self.totalQuantity fload, usually integer
- self.orderType MKT,LMT,STP,...
- self.lmtPrice limit price
- self.auxPrice when order type expects more than one price
- self.tif Time in Force (DAY, GTC, etc.))

Please also note, that only a tiny part or the order functionality has been tested. Be extremely careful when using more complex trades or combos in a live trading account.

Execution data can be filtered along the following criteria:

- ClientId int The API client which placed the order.
- AcctCode string The account to which the order was allocated to.
- Time string Time from which the executions will be returned yyyymmdd hh:mm:ss Only those executions reported after the specified time will be returned.
- Symbol string The instrument's symbol.
- SecType string The Contract's security's type (i.e. STK
- Exchange string The exchange at which the execution was produced.
- Side string The Contract's side (BUY or SELL)

Value

`cntr` returns a contract object

`ordr` returns an order object

`efltr` returns an executionFilter object

Examples

```
## Not run:
c1 = cntr(symbol="AAPL",exchange="SMART",secType="STK",currency="USD")
c2 = cntr(symbol="EUR",currency="USD",secType="CASH",exchange="IDEALPRO")
c3 = cntr(localSymbol = "ESZ4", exchange = "CME", secType="FUT")

## End(Not run)

## Not run:
oid  <- reqIds()
ord  <- ordr(orderId=oid,action="BUY",totalQuantity=7L,orderType="MKT",transmit = T)

## End(Not run)
```

---

order-management *placing, cancelling, updating and reporting orders*

Description

`placeOrder` submitts an order to a specified exchange or changes a submitted order — **nin-blocking**
`cancelOrder` discards an open order — **non-blocking**
`openOrder` gives an overview of currently open orders — **non blocking**
`completedOrders` lists all orders that recently were completed — **blocking**
`execReport` reports executed orders, commissions and more — **blocking**

Usage

```
placeOrder(orderId, cnt, ord)

cancelOrder(orderId)

openOrders()

execReport(efltr)

completedOrders()
```

Arguments

`orderId`	is a unique identifier to track for callbacks and reporting obtaineed by `reqIds()`
`cnt`	contract object defining the instrument and exchange
`ord`	order object detailling the order parameters
`efltr`	executionFilter object

Details

`placeOrder` submits a specific order to an exchange. After submission the callback 'open order' and 'order status' send status information until the order has been filled, canceled or not accepted. The order status can be accessed with `read.nb`. `placeOrder` is also used to change an open order. 'cancelOrder' discards an open order.
`openOrder` requests information about open orders in the same way that `placeOrder` through the callbacks 'open order' and 'order status'.
`execReport` creates a report of executed orders differentiated along the criteria defined in the `efltr` object that is given to the function as argument.
`completedOrders` provides all completed orders for the current client id only

Value

`placeOrder`, `cancelOrder`, `openOrder` are called for their side-effects only.
`execReport`, `completedOrders` returns (potentially) filtered list of executed orders.

`pprint` *pretty print python objects*

Description

pretty print python objects

Usage

```
pprint(l, v = 0)
```

Arguments

`l`	python object
`v`	nested level (internal)

Value

pretty printed python object

py2list *convert python object to list*

Description

basically just for order

Usage

```
py2list(l)
```

Arguments

l	python object

Value

python object as list

read.nb *callback data from non-blocking requests* — **non-blocking**

Description

Retrieves queued callback data written by callbacks after non-blocking requests. If src is provided, data is transformed to a more readable and processable format specific for each request.

Usage

```
read.nb(rq = NULL, src = NULL)
```

Arguments

rq	reqId to use for filter queue data. Not differentiated by rq if NULL.
src	text field denoting the request for which to transform. Not differentiating if NULL.

Value

returns accumulated data that was produced by callbacks.

Examples

```
## Not run:
cnt <- cntr(symbol="NVDA",exchange="SMART",secType="STK",currency="USD")

rid <- realTimeBars(cnt)
Sys.sleep(8)
cancelRealTimeBars(rid)
v1 <- read.nb(src="realTimeBars")


rid <- realTimeBars(cnt)
Sys.sleep(8)
cancelRealTimeBars(rid)
v2 <- read.nb(rq=rid)

## End(Not run)
```

`realtime-bars` *Request/ cancel five second live bar data* — **non-blocking**

Description

Request/cancel live 5 sec bar data subscription

Usage

```
realTimeBars(cnt, barSize = 5L, whatToShow = "TRADES", useRTH = "0")

cancelRealTimeBars(reqids)
```

Arguments

`cnt`	contract object
`barSize`	for future use; currently only 5sec bars are accepted
`whatToShow`	whatToShow, e.g. "TRADES", "MIDPOINT", "ASK", ...
`useRTH`	show bars within regular trading hours only
`reqids`	request IDs from subscription

Details

`realTimeBars` requests live bar data for a contract while `cancelRealTimeBars` stops the subscription.

Value

realTimeBars returns the reqId, which is needed to cancel the request later.

Examples

```
## Not run:
cnt <- cntr(symbol="NVDA",exchange="SMART",secType="STK",currency="USD")

rid <- realTimeBars(cnt)
Sys.sleep(8)
v1 <- read.nb(src="realTimeBars")
Sys.sleep(8)
cancelRealTimeBars(rid)
v2 <- read.nb(src="realTimeBars")

## End(Not run)
```

`reqIds` *reqIds - request new order id* — **blocking**

Description

Returns the session spanning valid order Id. This is persistent in TWS. Required to identify and dispatch callbacks.

Usage

```
reqIds()
```

Details

the orderIds persist in the TWS system. While it would be straightforward to,say, increment the order id after execution when there is a one client submitting orders, it becomes complicaed to ensure appropriate new oderIDs on oe client only. Do not confuse `reqIds` with `reqId`, the latter requests a requestId to track the callbacks on the client side. `reqIds`delivers] the next valid requestId

Value

valid order ID

`searchSymbols` *searchSymbols - find symbol and name* — **blocking**

Description

searching for contracts

Usage

```
searchSymbols(searchText)
```

Arguments

searchText	search text

Details

the search looks into IB'hosts fields of symbol, localSymbol, longName and returns each matchin contract

Value

"a list with all contracts that match based on 'symbol', 'localSymbol','longname'search results

Index