Package ‘IBrokers’

February 19, 2015

Type Package
Title R API to Interactive Brokers Trader Workstation
Version 0.9-12
Date 2014-09-22
Depends xts, zoo
Author Jeffrey A. Ryan
Maintainer Joshua M. Ulrich <josh.m.ulrich@gmail.com>
Description Provides native R access to Interactive Brokers Trader Workstation API.
License GPL-3
NeedsCompilation no
Repository CRAN
Date/Publication 2014-09-22 07:59:21

R topics documented:

IBrokers-package ......................................................... 2
.placeOrder ............................................................. 4
.twsIncomingMSG ....................................................... 5
calculateImpliedVolatility ........................................... 5
eWrapper ................................................................. 6
exerciseOptions ......................................................... 8
processMsg ............................................................... 10
reqAccountUpdates ..................................................... 11
reqContractDetails ..................................................... 12
reqCurrentTime ........................................................ 14
reqHistoricalData ...................................................... 15
reqIds ................................................................... 17
reqMktData ............................................................... 18
reqMktDepth ............................................................. 20
reqNewsBulletins ....................................................... 22
reqRealTimeBars ......................................................... 23
setServerLogLevel ....................................................... 25
IBrokers-package

R API to the Interactive Brokers Trader Workstation (TWS).

Description

This software is in no way affiliated, endorsed, or approved by Interactive Brokers or any of its affiliates. It comes with absolutely no warranty and should not be used in actual trading unless the user can read and understand the source.

IBrokers is a pure R implementation of the TWS API. At present it is only able pull data from the Interactive Brokers servers via the TWS. Future additions will include more API access, including live order handling, and better management across R sessions.

Possible real-time charting via the quantmod package may be incorporated into future releases.

Changes to move to version 0.1-0 have made this API implementation much more robust on all platforms. Many new error-checking calls have been incorporated, as well as a more reliable event-loop to capture the data from the TWS.

The underlying socket connections are pure R. This was a design decision to maximize cross-platform availability, as well as a recognition that historical data requests, or any requests while in a single threaded R session, must be non-threaded.

Recent additions include reqMktData to handle live market data from one or more symbols, reqMktDepth to capture market depth for one or more symbols, and reqRealTimeBars to receive 5 second real time bars. Each of these functions have been implemented with optional user defined callback handlers to allow for R code to interact with the API while receiving data from the TWS.

Please report any and all bugs/experiences to the maintainer so they can be corrected or incorporated into future versions.

Additionally, beta testers are needed to make this a viable alternative for IB-API interaction. Don’t be shy.

Details

Package: IBrokers
Type: Package
Version: 0.9-7
The current API methods supported are:

- `twsConnect`: Establish TWS connection
- `twsDisconnect`: Close TWS connection
- `isConnected`: Check connection
- `setServerLogLevel`: Set logging level
- `twsAccountUpdates`: Get Account Details
- `reqIds`: Request next available ID
- `reqCurrentTime`: The TWS server time in seconds since the epoch
- `reqHistoricalData`: Fetch historical data
- `reqMktData`: Receive real-time market data
- `reqMktDepth`: Receive real-time order book depth
- `reqRealTimeBars`: Receive 5 second OHLCVWC bar data

**Experimental support:**
- `placeOrder`: Place a live order to the TWS
- `cancelOrder`: Cancel a pending order on the TWS

**Author(s)**

Jeffrey A. Ryan
Maintainer: Joshua M. Ulrich <josh.m.ulrich@gmail.com>

**References**

Interactive Brokers: [www.interactivebrokers.com](http://www.interactivebrokers.com)

**Examples**

```r
## Not run:
IBrokersRef()  # IBrokers Reference Card in PDF viewer

tws <- twsConnect()  # make a new connection to the TWS
reqCurrentTime(tws)  # check the server's timestamp

contract <- twsEquity('IBKR','SMART','ISLAND')  # equity specification

reqHistoricalData(tws,contract)  # request historical data
twsDisconnect(tws)  # disconnect from the TWS
```

## End(Not run)
Description

Place or cancel an order to the TWS.

Usage

placeOrder(twsconn, Contract, Order)

cancelOrder(twsconn, orderId)

Arguments

twsconn A twsConnection object.
Contract A twsContract object.
Order A twsOrder object.
orderid A valid order id.

Details

As described by the official Interactive Brokers (tm) documentation. Caveat Emptor!!

Value

Called for its side effect of placing or cancelling an order on the TWS. This also returns the orderId used for placeOrder. An additional side-effect is that a variable .Last.orderId will be created or updated in the GlobalEnv as well.

Note

Orders via the API are quite complicated, or at least can be. It is strongly advised to only proceed with trading real money after one understands not only all the R code in this package, but the official API as well. If you are more comfortable clicking shiny buttons in a GUI, it is probably better that you keep clicking the buttons and not pretend to program.

Not for the faint of heart. All profits and losses related are yours and yours alone. If you don’t like it, write it yourself.

Author(s)

Jeffrey A. Ryan

References

See Also

twsContract twsOrder reqIds

Examples

## Not run:
tws <- twsConnect()
id <- reqIds(tws)

placeOrder(tws, twsSTK("AAPL"), twsOrder(id))
cancelOrder(id)

## End(Not run)

---

**Internal TWS-API MSG and ERR List**

Description

Internal List of MSG Codes and Undocumented (Experimental) Functions

---

**calculateImpliedVolatility**

*Calculate Option Values*

Description

Using the IB API, calculates the implied volatility or option price given parameters.

Usage

calculateImpliedVolatility(twsconn,
                        Contract,
                        optionPrice,
                        underPrice,
                        reqId = 1)

calculateOptionPrice(twsconn,
                      Contract,
                      volatility,
                      underPrice,
                      reqId = 1)
Arguments

- **twsconn**: A twsConnection object
- **Contract**: A twsContract object
- **optionPrice**: The option price from which to calculate implied volatility
- **vollatility**: The volatility from which to calculate price
- **underPrice**: The underlying price
- **reqId**: The request id

Details

Both calls will use the IB described method for calculation. See the official API for documentation.

Value

A numeric value corresponding to the request

Author(s)

Jeffrey A. Ryan

References


---

**eWrapper**  
**eWrapper Closure For Message Processing**

Description

Create an eWrapper closure to allow for custom incoming message management.

Usage

```
eWrapper(debug = FALSE, errfile=stderr())
eWrapper.data(n)
eWrapper.MktData.CSV(n=1)
eWrapper.RealTimeBars.CSV(n=1)
```

Arguments

- **debug**: should debugging be enabled
- **errfile**: where error messages are directed (stderr)
- **n**: number of contracts being watched
Details

IBrokers implements an eWrapper scheme similar to that provided by the official Java API.

The general idea is that each real-time data capture function must manage all incoming signals correctly, while allowing for the end user to create custom handlers for each specific event.

Internal to the reqRealTimeBars, reqMktData, and reqMktDepth functions is a single call to the CALLBACK routine passed to it. By default this is twsCALLBACK (see also). A standard argument to this callback is an eventWrapper — which is an instance of eWrapper.

eWrapper is an \texttt{R} closure that contains a list of functions to manage all incoming message type, as found in .twsIncomingMSG. Each message has a corresponding function in the eWrapper designed to handle the particular details of each incoming message type.

There is also an embedded environment in which data can be saved and retrieved via a handful of accessor functions mimicking the standard \texttt{R} tools.

The data environment is .Data, with accessor methods get.Data, assign.Data, and remove.Data. These methods can be called from the closure object ewrapper$.get.Data, ewrapper$.assign.Data, etc.

The basic eWrapper call simply produces a visually informative display of the incoming stream. E.g. bidSize data would be represented with a bidSize label, instead of the internal TWS code(s) returned by the TWS.

By creating an instance of an eWrapper, accomplished by calling it as a function call, one can then modify any or all the particular methods embedded in the object.

This allows for rapid customization, as well as a built in assurance that all incoming messages will be handled appropriately without additional programmer time and resources.

An example of this ability to modify the object is given in the ewrapper$.MktData$.CSV code. This object produces output designed to be space efficient, as well as easily read back into any R session as a standard CSV file.

Setting debug=NULL will cause empty function objects to be created within the eWrapper object returned. This object can be treated as a template to implement only the methods that are needed. By default, all functions silently return the entire message they would normally parse. This includes empty functions created by setting debug to NULL.

ewrapper$.data() allows for data states to be maintained from call to call, as an xts history of updates/messages is stored within the object. This is designed to minimize calling overhead by removing unneeded function calls from each message parsed.

Additional, but creating methods that update the internal environment of the eWrapper object, it is possible to maintain a snapshot of last k values for any field of interest. This is directly applicable to implementing an automated strategy from within a custom twsCALLBACK method.

Value

A list of functions [and optionally data] to be used for the eventWrapper argument to reqMktData and reqMktDepth
Note

It is possible to also attach data to the closure object, allowing for a single in-memory object to contain current top of book data. This is exemplified in the `eWrapper.MktData.CSV` code, and can be extended in the user's own direction.

Author(s)

Jeffrey A. Ryan

See Also

twsCALLBACK, processMsg

Examples

```r
myWrapper <- eWrapper()
str(myWrapper)

# remove tickPrice action
myWrapper$tickPrice <- function(msg, timestamp, file, ...) {
}

# add new tickPrice action
myWrapper$tickPrice <- function(msg, timestamp, file, ...) { cat("tickPrice",msg) }

# add new data into the object, and retrieve
myWrapper$assign.Data("myData", 1010)
myWrapper$get.Data("myData")

## Not run:
tws <- twsConnect()
reqMktData(tws, twsSTK("SBUX"))
reqMktData(tws, twsSTK("SBUX"), eventWrapper=myWrapper)
twsDisconnect(tws)

## End(Not run)
```

exerciseOptions Exercise Options Contracts

Description

Send message to exercise option contracts.
**Usage**

```java
eexerciseOptions(twsconn,
    contract,
    exerciseAction = 1,
    exerciseQuantity = 1,
    account = "",
    override = 0,
    tickerId = 1)
```

**Arguments**

- **twsconn**: A twsConnection object
- **contract**: A twsContract object
- **exerciseAction**: exercise=1 or lapse=2
- **exerciseQuantity**: number of contracts to exercise
- **account**: IB account [institutional orders]
- **override**: override system’s natural action. 0 for do not override, 1 for override
- **tickerId**: id for request

**Details**

Exercise option contract.

**Value**

Called for its side-effect.

**Note**

exch='SMART' is not valid in exerciseOptions calls. See the official API for further details.

**Author(s)**

Jeffrey A. Ryan

**References**

processMsg  

**Main TWS-API Event Manager**

**Description**

Function to manage all incoming messages from the TWS in a consistent manner.

This is used within the context of an event loop (often twsCALLBACK) and allows for custom processing by message type via the eWrapper argument.

**Usage**

```
processMsg(curMsg, con, eWrapper, timestamp, file, twsconn, ...)
```

**Arguments**

- **curMsg**: The current incoming message
- **con**: a socket connection from a twsConnection
- **eWrapper**: a functional closure with methods for each message
- **timestamp**: the timestamp format needed
- **file**: the file or connection to write to
- **twsconn**: the twsConnection object
- **...**: additional arguments to internal calls

**Details**

This is used internally within the context of a larger infinite listener/loop.

The basic process involves one or more requests to the TWS for data/action, followed by a call to twsCALLBACK. Inside of the CALLBACK is a loop that fetches the incoming message type, and calls processMsg at each new message.

processMsg internally is a series of if-else statements that branch according to a known incoming message type. The eWrapper object is a closure containing a data environment that is static and a collection of callback functions for each type of incoming data.

This eWrapper function can be defined at multiple points prior to the use within processMsg, to allow for access to data outside of the processMsg call, as well as facilitate custom handling in an efficient manner.

**Value**

Called for its side-effects.

**Note**

The entire mechanism (twsCALLBACK -> processMsg -> eWrapper) is modeled after the official API.
reqAccountUpdates

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers: http://wwwinteractivebrokers.com

See Also

twsCALLBACK, eWrapper

reqAccountUpdates Request Account Updates

Description

Request and view account details from Interactive Brokers

Usage

reqAccountUpdates(conn,
    subscribe = TRUE,
    acctCode = "1",
    eventWrapper = eWrapper(),
    CALLBACK=twsCALLBACK,
    ...
)

.reqAccountUpdates(conn, subscribe = TRUE, acctCode = "1")

twsPortfolioValue(x, zero.pos=TRUE, ...)

Arguments

conn A twsConnection object
subscribe subscribe (TRUE) or unsubscribe (FALSE)
acctCode an account description - not used for most accounts
eventWrapper message-level callback closure
CALLBACK main receiver loop, if any
x object to extract PortfolioValue from. See details.
zero.pos should PortfolioValue include zero positions?
... additional args
Details

By default, for non-FA accounts, this returns the current login’s account information. This main version returns a list of objects as returned by the TWS. .reqAccountUpdates sends the request to subscribe or cancel, but returns immediately. This is designed to be used within a larger custom callback routine, where the eventWrapper object passed to processMsg (see also) keeps trace of the portfolio updates in a consistent manner.

twsPortfolioValue extracts into a data.frame commonly used fields from all positions held. There are currently methods for the the default returned object of reqAccountUpdates.

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers API: http://www.interactivebrokers.com

Examples

```r
## Not run:
tws <- twsConnect()

reqAccountUpdates(tws)       # this will return a AccountUpdate object
   .reqAccountUpdates(tws)    # this will return immediately

   .reqAccountUpdates(tws, FALSE) # cancel the request
   cancelAccountUpdates(tws)   # the same

twsDisconnect(tws)

## End(Not run)
```

---

reqContractDetails  
Request Contract Details From TWS

Description

Returns an object (a list of class twsContractDetails objects) of IB contract details relating to a particular IB tradeable product.

Usage

```r
reqContractDetails(conn,  
                   Contract,  
                   reqId = "1",  
                   verbose = FALSE,  
                   eventWrapper = eWrapper(),  
                   CALLBACK = twsCALLBACK, ...)
```
Arguments

- **conn**: a valid twsConnection
- **Contract**: a valid twsContract
- **reqId**: a unique ID
- **verbose**: be verbose?
- **eventWrapper**: event callback closure
- **CALLBACK**: main callback loop
- **...**: be verbose?

Details

Returns a list of details for the product specified. See the TWS API for specifics at this point.

Value

A `twsContractDetails` object, or list of the same.

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers [http://www.interactivebrokers.com](http://www.interactivebrokers.com)

See Also

twsContract

Examples

```r
## Not run:
tws <- twsConnect()
reqContractDetails(tws, twsEquity("QQQQ"))

# retrieve all QQQQ contracts as a list
reqContractDetails(tws, twsOption(local="", right="", symbol="QQQQ"))
# retrieve only calls
reqContractDetails(tws, twsOption(local="", right="C", symbol="QQQQ"))
# retrieve only puts
reqContractDetails(tws, twsOption(local="", right="P", symbol="QQQQ"))

opt.details <- lapply(c("MSFT","AAPL"),
  function(x) {
  reqContractDetails(tws,
    twsOption(local="", right="", symbol=x))
  })
```
**Description**

Returns the current time from the TWS server, expressed as seconds since 1970-01-01 GMT.

**Usage**

```r
reqCurrentTime(twsconn)
```

**Arguments**

- `twsconn`: a valid tws connection object

**Value**

Seconds since 1970-01-01 GMT

**Author(s)**

Jeffrey A. Ryan

**References**

Interactive Brokers [www.interactivebrokers.com](http://www.interactivebrokers.com)

**Examples**

```r
## Not run:
tws <- twsConnect()
reqCurrentTime(tws)
## End(Not run)
```
reqHistoricalData

Request Historical Data From TWS

Description

Makes a request to the Interactive Brokers Trader Workstation (TWS), and returns an xts object containing the results of the request if successful.

Usage

reqHistoricalData(conn,  
  Contract,  
  endDateTime,  
  barSize = "1 day",  
  duration = "1 M",  
  useRTH = "1",  
  whatToShow = "TRADES",  
  timeFormat = "1",  
  tzone = ",",  
  verbose = TRUE,  
  tickerId = "1",  
  eventHistoricalData,  
  file)

reqHistory(conn, Contract, barSize, ...)

Arguments

conn a twsConnection object
Contract a twsContract
endDateTime end date/time for request. See details.
barSize bar size to retrieve
duration time span the request will cover
useRTH limited to regular trading hours
whatToShow type of data to be extracted
timeFormat POSIX style or seconds from 1970-01-01
tzone time zone of the resulting intraday series (if applicable)
verbose should progress be documented
tickerId a unique id to associate with the request
eventHistoricalData callback function to process data
file file to write data to
... args to pass to reqHistoricalData
Details

The `reqHistory` function is a simple wrapper to request maximal history from IB. It is meant to be used directly, or as a template for new wrappers.

All arguments should be character strings. Attempts will be made to coerce, but should not be relied upon.

The `endDateTime` argument must be of the form 'CCYYMMDD HH:MM:SS TZ'. If not specified the current time as returned from the TWS server will be used. This is the preferred method for backfilling data. The 'TZ' portion of the string is optional.

Legal `barSize` settings are technically ‘1 secs’, ‘5 secs’, ‘15 secs’, ‘30 mins’, ‘1 min’, ‘2 mins’, ‘3 mins’, ‘5 mins’, ‘15 mins’, ‘30 mins’, ‘1 hour’, ‘1 day’, ‘1 week’, ‘1 month’, ‘3 months’, and ‘1 year’. They must be specified exactly and there is no guarantee from the API that all will work for all securities or durations.

The duration string must be of the form ‘n S’ where the last character may be any one of ‘S’ (seconds), ‘D’ (days), ‘W’ (weeks), ‘M’ (months), and ‘Y’ (year). At present the limit for years is 1.

`userTh` takes either ‘1’ or ‘0’, indicating the request to return only regular trade hour data, or all data, respectively.

`whatToShow` can be any one of the following, though depending on the overall request it may not succeed. ‘TRADES’, ‘MIDPOINT’, ‘BID’, ‘ASK’, ‘BID_ASK’.

`timeFormat` should simply be left alone. :D

eventHistoricalData accepts a user function to process the raw data returned by the TWS. This consists of a character vector that includes the first five elements of header information, with the fifth element specifying the number of rows in the results set. Passing NULL to eventHistoricalData will return the raw character vector. If nothing is specified, an xts object is returned.

The `eventHistoricalData` function, if any, is called after all data has been received by the client.

The `file` argument calls `write.table` to produce output suitable to reading in by `read.csv`. The `file` argument is passed to the `write.table` call, and if an empty string will return the output to the console.

The `hasGaps` column is converted automatically from (true,false) to 1 or 0, respectively.

Value

Returns an xts object containing the requested data, along with additional information stored in the objects `xtsAttributes`, unless `callback` or `file` is specified.

Note

The rules for historical data requests are somewhat vague. Not all symbols have data, and those that do may only be available with specific combinations of `barSize` and duration arguments. At present the only way to know is to try the combination in question.

There is a strictly enforced 10 seconds between request pacing rule implemented by the TWS. Keep this in mind. IBrokers currently does not manage this for the user via `reqHistoricalData`, though `reqHistory` does.
reqIds

Author(s)
Jeffrey A. Ryan

References
Interactive Brokers www.interactivebrokers.com

See Also
twsContract, twsConnect

Examples

```r
## Not run:
tws <- twsConnect()
contract <- twsEquity('QQQ', 'SMART', 'ISLAND')

# by default retrieves 30 days of daily data
reqHistoricalData(tws, Contract(contract))

# by default retrieves a year of 1 minute bars
Sys.sleep(10) # mandatory 10s between request to avoid IB pacing violation
reqHistory(tws, Contract(contract))

## End(Not run)
```

reqIds Request Next Valid Id

Description
Get the next valid order ID for use with the TWS.

Usage
```
reqIds(conn, numIds = 1)
```

Arguments
conn a valid twsConnection object of class twsconn.
umIds currently ignored by the TWS.

Details
twsconn objects maintain the next valid id inside of the object, returning the current id, and incrementing by 1 with each call to reqIds.
For twsconn objects, reqIds and .reqIds results are identical.
reqMktData

Value

A character representation of the next numeric ID.

Note

The TWS will keep track of order ids across connection ids and sessions. The values may be reset only as outlined by the official TWS documentation. IBrokers simply records and manages the data as received from the TWS upon initial connection. Each connection id will have a different order id associated with it.

Author(s)

Jeffrey A. Ryan

---

**reqMktData**

Request Market Data Feed from TWS

---

Description

Allows for streaming market data to be handled in R.

Usage

```r
reqMktData(conn,
            Contract,
            tickGenerics = "100,101,104,106,165,221,225,236",
            snapshot = FALSE,
            tickerId = "1",
            timestamp = "%Y%m%d %H:%M:%OS",
            playback = 1,
            file = "",
            verbose = TRUE,
            eventWrapper = eWrapper(),
            CALLBACK = twsCALLBACK, ...)
```

cancelMktData(conn,tickerId)

Arguments

- conn: a valid twsConnection or twsPlayback connection
- Contract: twsContract object(s) requested data for
- tickGenerics: a common delimited string of generic tick types
- snapshot: should snapshot data be returned
- tickerId: the ticker id to associate with the returned data
- timestamp: include R time stamps
reqMktData

playback  playback speed adjustment
file      passed to internal cat calls. See associated help
verbose   print diagnostics?
eventWrapper eWrapper object
CALLBACK main reciever callback
...       additional args

Details

This function provides \texttt{R} level access to market data streams as returned by the TWS API. The Interactive Brokers documentation should be reference for the exact meaning of the returned data.

timeStamps is unique to the \texttt{R} API in that each incoming signal will be marked with a (potentially) unique timestamp. Alternatively it is possible to pass a formatting string for use in \texttt{format(Sys.time())}. To suppress the time stamp set the argument to \texttt{NULL}. This is not sent by the TWS - merely prepended to the output by \texttt{R}.

Callbacks, via CALLBACK and eventWrapper are designed to allow for \texttt{R} level processing of the real-time data stream.

Each message recieved (each update to the market data) will invoke one the appropriately names eWrapper callback, depending on the message type. By default when nothing is specified, the code will call the default method for printing the results to the screen via \texttt{cat}.

Note that the use of the argument file will be passed to these cat calls, and therefore it will be possible to use the functionability of \texttt{cat} directly - e.g. piping output or writing to a connection. The simplest use of file would be to specify the name of a file to append the output of the stream to.

The CALLBACK argument is used for more control of the incoming results. This requires user-level error checking as well as TWS API interaction. It is here for advanced use and until documented should be left alone.

Value

The real-time market data from the TWS.

Note

As \texttt{R} is single threaded - this request will run until interrupted by an error or by user action. Both will clean up after themselves when appropriate.

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers API: \url{http://individuals.interactivebrokers.com/php/apiguide/apiguide.htm}
See Also
twscallback, eWrapper, twsConnect, twsContract

Examples

```r
## Not run:
tws <- twsConnect()
contract <- twsEquity("QQQQ","SMART","ISLAND")
reqMktData(tws, contract)

# write to an open file connection
fh <- file("out.dat",open='a')
reqMktData(tws, contract, file=fh)
close(fh)

## End(Not run)
```

---

`reqMktDepth`  
Request Market Depth Feed from TWS

**Description**

Allows for streaming market depth (order book) data to be handled in R.

**Usage**

```r
reqMktDepth(conn,
            Contract,
            tickerId = "1",
            numRows = "20",
            timeStamp = TRUE,
            playback = 1,
            file = "",
            verbose = TRUE,
            eventWrapper = eWrapper(),
            CALLBACK = twSCALLBACK, ...)
```

`cancelMktDepth(conn,tickerId)`

**Arguments**

- `conn`  
a valid twsConnection connection
- `Contract`  
twsContract object(s) requested data for
- `tickerId`  
the ticker id to associate with the returned data
- `numRows`  
depth of book
- `timeStamp`  
include R time stamps
reqMktDepth

playback  playback speed adjustment
file      passed to internal cat calls. See associated help.
verbose   print diagnostics?
eventWrapper     callback closure
CALLBACK main receiver loop
...    additional args

Details

This function provides R level access to book data as returned by the TWS API. The Interactive Brokers documentation should be reference for the exact meaning of the returned data.

timeStamps is unique to the R API in that each incoming signal will be marked with a (potentially) unique timestamp. Alternatively it is possible to pass a formatting string for use in format(Sys.time()). To suppress the time stamp set the argument to NULL.

Callbacks, via eventUpdateMktDepth, eventUpdateMktDepthL2, or CALLBACK are designed to allow for R level processing of the real-time data stream.

The first two correspond to actions based upon the actual signal recived. These may be user-defined functions taking the appropriate arguments. Each message recieved (each update to the market depth) will invoke one of these callbacks. By default when nothing is specified, the code will call the default method for printing the results to the screen via cat.

Note that the use of the argument file will be passed to these cat calls, and therefore it will be possible to use the functionality of cat directly - e.g. piping output or writing to a connection. The simplest use of file would be to specify the name of a file to append the output of the stream to.

The CALLBACK argument is used for more control of the incoming results. This requires user-level error checking as well as TWS API interaction. It is here for advanced use and until documented should be left alone.

Value

The book depth.

Note

As R is single threaded - this request will run until interrupted by an error or by user action. Both will clean up after themselves when appropriate.

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers API: http://individuals.interactivebrokers.com/php/apiguide/apiguide.htm
See Also
twsConnect, twsContract

Examples

```r
## Not run:
tws <- twsConnect()
contract <- twsEquity("QQQ","SMART","ISLAND")
reqMktDepth(tws, contract)

# write to a file
reqMktDepth(tws, contract, file='out.dat')

## End(Not run)
```

---

`reqNewsBulletins` *Subscribe or Unsubscribe To News Bulletins*

**Description**

Subscription start and end methods for the API.

**Usage**

```r
reqNewsBulletins(twscnn, allMsgs=TRUE)
cancelNewsBulletins(twscnn)
```

**Arguments**

- `twscnn`: A `twsConnection` object
- `allMsgs`: Should all existing bulletins be returned (TRUE), or just new ones?

**Details**

Calling `reqNewsBulletins` will start a subscription via the API. This will continue and incoming messages will be handled by `eWrapper` `updateNewBulletin` method. Bulletins are cancelled by calling the cancel version.

**Value**

Called for its side-effects.

**Note**

This is not “news” per se, it is a subscription to the API bulletins.
reqRealTimeBars

Author(s)
Jeffrey A. Ryan

References

reqRealTimeBars Request Real Time Bars from TWS

Description
Allows for streaming real-time bars to be handled in R

Usage
reqRealTimeBars(conn,
            Contract,
            whatToShow = "TRADES",
            barSize = "5",
            useRTH = TRUE,
            playback = 1,
            tickerId = "1",
            file = "",
            verbose = TRUE,
            eventWrapper=eventWrapper(),
            CALLBACK=twsCALLBACK,
            ...
)

cancelRealTimeBars(conn, tickerId)

Arguments
conn
a valid twsConnection or twsPlayback object
Contract
twsContract object(s) requested
tickerId
the ticker id to associate with the returned bars
whatToShow
what to show
barSize
bar size - currently on 5 secs is TWS supported
playback
playback speed adjustment
useRTH
regular trading hours (logical)
file
passed to internal cat calls. See associated help.
verbose
print diagnostics
eventWrapper
eventWrapper object
CALLBACK
main reciever callback
...
additional args to callback
Details

This function provides R level access to real time (5 second) bars returned by the TWS API. The Interactive Brokers documentation should be reference for the exact meaning of the returned data.

If the `conn` is a connection of data to be played back all other arguments are ignores, except for `playback`, which is a multiplier of the bar size in seconds. To force all data to be read without pause set this to 0.

Callbacks, via `eventRealTimeBars` and `CALLBACK` are designed to allow for R level processing of the real-time data stream.

eventWrapper allows for direct manipulation of the actual signal received. These may be user-defined functions taking the appropriate arguments. Each message received (each new bar) will invoke one of this callback. By default when nothing is specified, the code will call the default method for printing the results to the screen via 'cat'.

Note that the use of the argument ‘file’ will be passed to these 'cat' calls, and therefore it will be possible to use the functionality of ‘cat’ directly - e.g. piping output or writing to a connection. The simplest use of file would be to specify the name of a file, or open connection to append the output of the stream to.

The ‘CALLBACK’ argument is used for more control of the incoming results. This requires user-level error checking as well as TWS API interaction. It is here for advanced use and until documented should be left alone.

Value

The real-time bar data requested.

Note

As R is single threaded - this request will run until interrupted by an error or by user action. Both will clean up after themselves when appropriate.

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers TWS API [http://individualsinteractivebrokerscom/php/apiguide/ apiguide.htm](http://individualsinteractivebrokerscom/php/apiguide/ apiguide.htm)

See Also

twsConnect, twsContract, eWrapper

Examples

```r
## Not run:
tws <- twsConnect()
contract <- twsEquity("QQQ","SMART","ISLAND")
reqRealTimeBars(tws, contract)
```
setServerLogLevel

# write to an open file connection
fh <- file('out.dat', open='a')
reqRealTimeBars(tws, contract, file=fh)
close(fh)

## End(Not run)

---

**setServerLogLevel**  
**Enable API Logging Via TWS**

**Description**
Set level of API logging to be done by TWS.

**Usage**
setServerLogLevel(conn, logLevel = 2)

**Arguments**
- conn: a valid twsConnection
- logLevel: an integer from 1 to 5

**Details**
Calling this function will set the logging level for the current connection according to the following table:

1. 1:SYSTEM (least detail)
2. 2:ERROR (default)
3. 3:WARNING
4. 4:INFORMATION
5. 5:DETAIL (most detail)

See TWS documentation for further details.

**Value**
This function is called for its side-effects.

**Note**
The online documentation warns of performance overhead when setting logLevel=5.

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

**References**


---

**Description**

twsCALLBACK is the primary function that is called after a request for data is sent. Within this call messages are received from the TWS, processed, and further actions can be handled.

**Usage**

```
twsCALLBACK(twsCon, eWrapper, timestamp, file, playback = 1, ...)
```

**Arguments**

- `twsCon`: a `twsConnection` object
- `eWrapper`: a closure created by `eWrapper()`
- `timestamp`: a logical indicating if timestamps should be created
- `file`: the file or connection to write to
- `playback`: is this a live or playback connection
- `...`: additional arguments to internal calls

**Details**

This function is used as the primary management tool within all data calls built into IBrokers. It works as is, or can be modified to manage unique data and trading requirements.

The general logic of the function is to receive the header to each incoming message from the TWS. This then gets passed to the `processMsg` function, along with the `eWrapper` object.

The `eWrapper` object can maintain state data (prices), and has functions for managing all incoming message types from the TWS.

Once the `processMsg` call returns, another cycle of the infinite loop occurs.

If the `eWrapper` object is used to maintain state information, it is possible to access this information from outside of the `processMsg` call, and thus be able to apply trade logic based upon the data acquired from the TWS.

An example will soon be available in the vignettes included in the package.

**Value**

No value is returned. This function is called for its side effects.
twsConnect

Author(s)

Jeffrey A. Ryan

See Also
eWrapper

twsConnect

Establish, Check or Terminate a Connection to TWS or IBG

Description

Functions to initiate, check, or disconnect from the Trader Workstation (TWS) or IB Gateway (IBG).

Usage

twsConnect(clientId = 1, host = 'localhost',
port = 7496, verbose = TRUE, timeout = 5,
filename = NULL, blocking=.Platform$OS.type=="windows")
ibgConnect(clientId = 1, host = 'localhost',
port = 4001, verbose = TRUE, timeout = 5,
filename = NULL, blocking=.Platform$OS.type=="windows")
twsDisconnect(twsconn)

isConnected(twsconn)

is.twsConnection(x)

is.twsPlayback(x)

Arguments

- clientId: the unique client id to associate with
- host: the host server
- port: the port that the TWS is listening on
- verbose: should the connection attempt be verbose
- timeout: length in seconds before aborting attempt
- filename: file containing recorded TWS data
- blocking: should a blocking connection be established. See details.
- twsconn: a valid twsConnection object
- x: a connection to be checked
**Details**

Returns a `twsConnection` object for use in subsequent TWS API calls. Attempting to create another connection to the server with the same clientId will result in an error.

If `filename` is set to a file containing data recorded in the standard TWS format - calls using this connection will playback the recorded data.

While the `IBrokers` package is fully cross-platform, the behavior of sockets varies by operating system implementation. In general, setting `blocking=TRUE` on Windows (the default on Windows) results in more consistent and reliable connections. This option is only exposed to enable debugging of platform differences and optimization - and is not intended to be altered except in those cases.

**Value**

A `twsconn` object.

**Note**

While it is not strictly required to disconnect via `twsDisconnect` it is probably advisable.

If not set options(digits.secs=6) will be called internally to properly represent on screen the R based timestamps.

**Author(s)**

Jeffrey A. Ryan

**References**

Interactive Brokers: [www.interactivebrokers.com](http://www.interactivebrokers.com)

**Examples**

```r
## Not run:
tws <- twsConnect()
twsDisconnect(tws)
## End(Not run)
```

---

**twsConnectionTime**

* `TWS API Utility Functions`

**Description**

General API utility functions.

**Usage**

```r
twsConnectionTime(con)
serverVersion(con)
```
twsContract

Arguments

con a twsConnection object

Details

This is simply extracted from the twsConnection object. No API request is made.

Value

The requested value.

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers LLC http://www.interactivebrokers.com

See Also

twsConnect

Examples

## Not run:
twsConnectionTime(con)
serverVersion(con)

## End(Not run)

---

twsContract Create a twsContract

Description

Create, test, and coerce a twsContract for use in API calls.

Usage

twsContract(conId,
symbol,
sectype,
exch,
primary,
expiry,
strike,
currency,
right,
local,
multiplier,
combo_legs_desc,
comboleg,
include_expired,
secIdType = "",
secId = ""
)

is.twsContract(x)
as.twsContract(x, ...)

Arguments

conId the IB contract ID
symbol the IB symbol requested
sectype the security type
exch the requested exchange
primary the primary exchange of the security
expiry the expiration date
strike the strike price
currency the requested currency
right the requested right
local the local security name
multiplier the contract multiplier
combo_legs_desc not implemented yet
comboleg not implemented yet
include_expired should expired contracts be included
secIdType unique identifier for secIdType
secId security identifier: ISIN, CUSIP, SEDOL, RIC
x object to test or coerce
... additional arguments

Details

These are directly from the TWS API. See that help until I can find time to fill in this one.

More useful for specific requests are twsEquity, twsOption, twsBond, twsFuture, and twsCurrency.

Value

A twsContract object.
twsCurrency

**Author(s)**

Jeffrey A. Ryan

**References**

Interactive Brokers: [www.interactivebrokers.com](http://www.interactivebrokers.com)

**See Also**

reqHistoricalData

**Examples**

```r
contract <- twsContract(0,"AAPL","STK","SMART","ISLAND",
"","
0.0","USD","","","",NULL,NULL,"0")
```

---

**twsCurrency**  
*Create a twsCurrency*

**Description**

Create a twsCurrency for use in API calls.

**Usage**

```r
twsCurrency(symbol,
  currency='USD',
  exch='IDEALPRO',
  primary='',
  strike='0.0',
  right='',
  local='',
  multiplier='',
  include_expired='0',
  conId=0)
```

**Arguments**

- **symbol** the IB symbol requested
- **currency** the requested currency
- **exch** the requested exchange
- **primary** the primary exchange of the security
- **strike** the strike price
- **right** the requested right
- **local** the local security name
multiplier the contract multiplier
include_expired should expired contracts be included
conId contract ID

Details
A wrapper to twsContract to make ‘currency/FX’ contracts easier to specify.
twsCASH is an alias.

Value
A twsContract object.

Author(s)
Jeffrey A. Ryan

References
Interactive Brokers: www.interactivebrokers.com

See Also
reqHistoricalData, twsContract

Examples
currency <- twsCurrency("EUR")

twsEquity Create a twsEquity

description
Create a twsEquity for use in API calls.

Usage
twsEquity(symbol,
exch="SMART",
primary,
strike='0.0',
currency='USD',
right='',
local='',
multiplier='',
include_expired='0',
conId=0)
Arguments

- **symbol**: the IB symbol requested
- **exch**: the requested exchange (defaults to ‘SMART’)
- **primary**: the primary exchange of the security
- **strike**: the strike price
- **currency**: the requested currency
- **right**: the requested right
- **local**: the local security name
- **multiplier**: the contract multiplier
- **include_expired**: should expired contracts be included
- **conId**: contract ID

Details

A wrapper to `twsContract` to make ‘equity’ contracts easier to specify.
`twsSTK` is an alias.

Value

A `twsContract` object.

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers: [www.interactivebrokers.com](http://www.interactivebrokers.com)

See Also

`reqHistoricalData`, `twsContract`

Examples

```r
equity <- twsEquity("AAPL","SMART","ISLAND")
```
twsFuture

Create a twsFuture Contract

Description

Create a twsFuture contract for use in API calls.

Usage

twsFuture(symbol, 
exch, 
expiry, 
primary='', 
currency='USD', 
right='', 
local='', 
multiplier='', 
include_expired='0', 
conId=0)

Arguments

- **symbol**: the IB symbol requested
- **exch**: the requested exchange
- **expiry**: the requested contract expiration
- **primary**: the primary exchange of the security
- **currency**: the requested currency
- **right**: the requested right
- **local**: the local security name
- **multiplier**: the contract multiplier
- **include_expired**: should expired contracts be included
- **conId**: contract ID

Details

A wrapper to twsContract to make ‘futures’ contracts easier to specify.
twsFUT is an alias.

Value

A twsContract object.
twsOption

Author(s)

Jeffrey A. Ryan

References

Interactive Brokers: www.interactivebrokers.com

See Also

reqHistoricalData, twsContract

Examples

future <- twsFuture("NQ","GLOBEX","200803")

twsOption Create a twsContract for Options

Description

Create a twsContract for use in API calls.

Usage

twsOption(local,
    expiry="",
    strike="",
    right="",
    exch="SMART",
    primary="",
    currency='USD',
    symbol='',
    multiplier="100",
    include_expired='0',
    conId=0)

Arguments

    local the IB symbol requested
    expiry option expiration CCYYMM [optional]
    strike the strike price [optional]
    right the requested right - ‘C’,*CALL*, ‘P’,* or *PUT* [optional]
    exch the requested exchange [optional, defaults to SMART]
    primary the primary exchange of the security [optional]
    currency the requested currency [defaults to USD]
symbol  the security name [optional]
multiplier the contract multiplier
include_expired should expired contracts be included [defaults to “0” (false)]
conId    contract ID

Details
A wrapper to twsContract to make ‘option’ contracts easier to specify.
Some of the optionable parameters are contingent on the request being made. Refer to the TWS documentation for details.
twsOPT is an alias.

Value
A twsContract object.

Note
Option contracts on the TWS have certain rules which are different than standard data requests.
The local symbol is required. This can be found on the main TWS screen under contract details, or via the web at wwwinteractivebrokers.com
Since the local symbol is required, all other values are redundant. It is best to simply specify the local name and let the TWS manage the lookup.
The expiry needs to be either of class Date to be coerced to a string of format ‘CCYYMM’, or provided in that format.
Historical requests cannot be for a barSize='1 D’ or less frequent.
barSize must be "1 min” per Interactive Brokers API.

Author(s)
Jeffrey A. Ryan

References
Interactive Brokers: wwwinteractivebrokers.com

See Also
reqMktData, twsContract

Examples
```r
ton <- twsOption("QQQAS", expiry="200901", strike="45.0", right="C")```
Create twsOrder Object

Description

Create twsOrder object for placeorder API call.

Usage

twsOrder(orderId,
    action = "BUY",
    totalQuantity = "10",
    orderType = "LMT",
    lmtPrice = "0.0",
    auxPrice = "0.0",
    tif = "",
    outsideRTH = "0",
    openClose = "0",
    origin = .twsOrderID$CUSTOMER,
    ocaGroup = "",
    account = "",
    orderRef = "",
    transmit = TRUE,
    parentId = "0",
    blockOrder = "0",
    sweepToFill = "0",
    displaySize = "0",
    triggerMethod = "0",
    hidden = "0",
    discretionaryAmt = "0.0",
    goodAfterTime = "",
    goodTillDate = "",
    faGroup = "",
    faMethod = "",
    faPercentage = "",
    faProfile = "",
    shortSaleSlot = "0",
    designatedLocation = .twsOrderID$EMPTY_STR,
    ocaType = "0",
    rule80A = "",
    settlingFirm = "",
    clearingAccount = "",
    clearingIntent = "",
    allOrNone = "0",
    minQty = "",
    percentOffset = "",
    eTradeOnly = "0",
)
firmQuoteOnly = "0",
nbboPriceCap = "",
auctionStrategy = "0",
startingPrice = "",
stockRefPrice = "",
delta = "",
stockRangeLower = "",
stockRangeUpper = "",
overridePercentageConstraints = "0",
volatility = "",
volatilityType = "",
deltaNeutralOrderType = "",
deltaNeutralAuxPrice = "",
continuousUpdate = "0",
referencePriceType = "",
trailStopPrice = "",
basisPoints = "",
basisPointsType = "",
scaleInitLevelSize = "",
scaleSubsLevelSize = "",
scalePriceIncrement = "",
notHeld = false,
algoStrategy = "",
algoParams = NULL,
whatIf = false,
clientId = "",
permId = "")

Arguments

**orderId** The id for the order. Use reqIds.

**action** Identifies the side. (BUY, SELL, SSHORT)

**totalQuantity** Order quantity.

**ordertype** Order type. (MKT, MKTCLS, LMT, LMTCLS, PEGMKT, SCALE, STP, STPLMT, TRAIL, REL, VWAP, TRAILLIMIT)

**lmtPrice** The LIMIT price for LMT, STPLMT and REL ordertype

**auxPrice** The STOP price for STPLMT (stop-limit) orders, and the offset for REL (relative) orders

**tif** Time in force. (DAY, GTC, IOC, GTD)

**outsideRTH** Allow orders to trigger outside of regular trading hours.

**openclose** Specify whether order is open or close only. (Institutional Accounts Only)

**origin** The order origin. 0=customer, 1=firm (Institutional Accounts Only)

**ocaGroup** Identifies OCA group.

**account** The account (Institutional Accounts Only)

**orderRef** The order reference (Institutional Accounts Only)
transmit
Specify whether the order is transmitted to the TWS. If FALSE, order is created but not sent. (not implemented)

parentId
The orderId of the parent order, used for bracket and auto trailing stop orders.

blockOrder
ISE block order?

sweepToFill
Sweep to fill order?

displaySize
Publicly disclosed order size for Iceberg orders.

triggerMethod
How should simulated orders be triggered. Valid values are 0-8. See the official API for details.

hidden
Hide order on ISLAND?

discretionaryAmt
Amount off limit for discretionary orders.

goodAfterTime
Trades Good After Time: YYYYMMDD hh:mm:ss or ""

goodTillDate
Trades Good Till Date: YYYYMMDD hh:mm:ss or ""

faGroup
NA

faMethod
NA

faPercentage
NA

faProfile
NA

shortSaleSlot
1 or 2

designatedLocation
Only when shortSaleSlot=2

ocaType
Cancel on Fill with Block = 1 Reduce on Fill with Block = 2 Reduce on Fill without Block = 3

rule80A
Valid values: I, A, W, J, U, M, K, Y, N. See API.

settlingFirm
(Institutional Only)

clearingAccount
IBExecution customers only.

clearingIntent
IBExecution customers only.

allOrNone
yes=1, no=0

minQty
Minimum quantity order type.

percentOffset
Percent offset for REL (relative) orders.

eTradeOnly
Trade with electronic quotes. yes=1, no=0.

firmQuoteOnly
Trade with firm quotes. yes=1, no=0.

nbboPriceCap
The maximum Smart order distance from the NBBO.

auctionStrategy
BOX only. See API.

startingPrice
BOX only. See API.

stockRefPrice
The stock reference price. VOL orders. See API.

delta
BOX only. See API.

stockRangeLower
See API.
stockRangeUpper
  See API.
overridePercentageConstraints
  See API.
volatility
  See API.
volatilityType
  See API.
deltaNeutralOrderType
  See API.
deltaNeutralAuxPrice
  See API.
continuousUpdate
  See API.
referencePriceType
  See API.
trailStopPrice
  For TRAILLIMIT orders only.
basisPoints
  EFP orders only.
basisPointsType
  EFP orders only.
scaleInitLevelSize
  For Scale orders. See API.
scaleSubsLevelSize
  For Scale orders. See API.
scalePriceIncrement
  For Scale orders. See API.
notHeld
  See API and guess.
algoStrategy
  See API and guess.
algoParams
  See API and guess.
whatIf
  Use to request pre-trade commissions and margin information. TRUE/FALSE
clientId
  Id of the client that placed the order.
permId
  TWS id used to identify orders. Constant over a session.

Details

Read the API documentation, code, and experiment with the paper accounts. And good luck!

Value

Called for its side-effects.

Note

Documentation is far from complete on this topic. Experiment and share your experiences.

Author(s)

Jeffrey A. Ryan
twsScannerSubscription

References


See Also

placeOrder

twsScannerSubscription

Create ScannerSubscription

Description

Create an object for use with reqScannerSubscription and .reqScannerSubscription.

Usage

```r
trwscannersubscription(numberOfRows = -1,
instrument = "",
locationCode = "",
scanCode = "",
abovePrice = "",
belowPrice = "",
aboveVolume = "",
averageOptionVolumeAbove = "",
marketCapAbove = "",
marketCapBelow = "",
moodyRatingAbove = "",
moodyRatingBelow = "",
spRatingAbove = "",
spRatingBelow = "",
maturityDateAbove = "",
maturityDateBelow = "",
couponRateAbove = "",
couponRateBelow = "",
excludeConvertible = "",
scannerSettingPairs = "",
stockTypeFilter = "")
```

Arguments

- `numberOfRows`: Number of rows of scanner results returned
- `instrument`: A character string of STK, ...
- `locationCode`: A character string of STK.NA, STK.US, STK.US.MAJOR, ...
- `scanCode`: One of the available scans. See details
abovePrice  Price to filter above
belowPrice  Price to filter below
aboveVolume  Volume to filter above
averageOptionVolumeAbove
  Average option volume above this
marketCapAbove  Market cap to filter above
marketCapBelow  Market cap to filter below
moodyRatingAbove
  Moody rating to filter above
moodyRatingBelow
  Moody rating to filter below
spRatingAbove  S&P rating to filter above
spRatingBelow  S&P rating to filter below
maturityDateAbove
  Maturity date to filter above
maturityDateBelow
  Maturity date to filter below
couponRateAbove
  Coupon rate to filter above
couponRateBelow
  Coupon rate to filter below
excludeConvertible
  ?
scannerSettingPairs
  ?
stockTypeFilter
  "ALL"?

Details

By necessity, design, or otherwise - scanner data is difficult to correctly use at the API level. The valid values and some small examples are returned by the API using the related reqScannerParameters function. The XML returned by that call isn’t very clear in its value or purpose though.

Value

A (potentially) valid twsScannerSubscription object for reqScannerSubscription calls.

Note

Further documentation will be forthcoming. Users are encouraged to email use cases to make for better documentation.

Author(s)

Jeffrey A. Ryan
References


See Also

reqScannerSubscription,

Examples

```r
scnr <- twsScannerSubscription(numberOfRows=10,
                                instrument="STK",
                                locationCode="STK.US.MAJOR",
                                scanCode="TOP_PERC_GAIN",
                                aboveVolume=0,
                                marketCapAbove=1e8,
                                scannerSettingPairs="Annual,true",
                                stockTypeFilter="ALL")

scnr
```
Index

*Topic textasciitildedekwd1
  .placeOrder, 4
twsOrder, 37
*Topic textasciitildedekwd2
  .placeOrder, 4
twsOrder, 37
*Topic class
twsScannerSubscription, 41
*Topic misc
calculateImpliedVolatility, 5
exerciseOptions, 8
reqAccountUpdates, 11
reqNewsBulletins, 22
twsCALLBACK, 26
*Topic package
IBrokers-package, 2
*Topic utilities
twsIncomingMSG, 5
eWrapper, 6
processMsg, 10
reqAccountUpdates, 11
reqContractDetails, 12
reqCurrentTime, 14
reqHistoricalData, 15
reqIds, 17
reqMktData, 18
reqMktDepth, 20
reqRealTimeBars, 23
setServerLogLevel, 25
twsCALLBACK, 26
twsConnect, 27
twsConnectionTime, 28
twsContract, 29
twsCurrency, 31
twsEquity, 32
twsFuture, 34
twsOption, 35
.lastRequest (.twsIncomingMSG), 5
.placeOrder, 4
.reqAccountUpdates (reqAccountUpdates), 11
.reqIds (reqIds), 17
twsERR (.twsIncomingMSG), 5
twsIncomingMSG, 5
twsOutgoingMSG (.twsIncomingMSG), 5
as.twsContract (twsContract), 29
calculateImpliedVolatility, 5
calculateOptionPrice
  (calculateImpliedVolatility), 5
cancelAccountUpdates
  (reqAccountUpdates), 11
cancelHistoricalData
  (reqHistoricalData), 15
cancelMktData (reqMktData), 18
cancelMktDepth (reqMktDepth), 20
cancelNewsBulletins (reqNewsBulletins), 22
cancelOrder (.placeOrder), 4
cancelRealTimeBars (reqRealTimeBars), 23
eWrapper, 6, 11, 20, 24, 27
exerciseOptions, 8
ibgConnect (twsConnect), 27
IBrokers (IBrokers-package), 2
IBrokers-package, 2
IBrokersRef (IBrokers-package), 2
is.twsConnection (twsConnect), 27
is.twsContract (twsContract), 29
is.twsPlayback (twsConnect), 27
isConnected (twsConnect), 27
placeOrder, 4
placeOrder (.placeOrder), 4
processMsg, 8, 10
replaceFA (.twsIncomingMSG), 5
reqAccountUpdates, 11
reqContractDetails, 12
twSTK (twsEquity), 32
reqCurrentTime, 14
reqExecutions (.twsIncomingMSG), 5
reqHistoricalData, 15, 31–33, 35
reqHistory (reqHistoricalData), 15
reqIds, 5, 17
reqManagedAccts (.twsIncomingMSG), 5
reqMktData, 18, 36
reqMktDepth, 20
reqNewsBulletins, 22
reqOpenOrders (.twsIncomingMSG), 5
reqRealTimeBars, 23
reqScannerParameters (.twsIncomingMSG), 5
reqScannerSubscription, 43
reqScannerSubscription (.twsIncomingMSG), 5
requestFA (.twsIncomingMSG), 5

serverVersion (twsConnectionTime), 28
setServerLogLevel, 25

twsBAG (.twsIncomingMSG), 5
twsCALLBACK, 8, 11, 20, 26
twsCASH (twsCurrency), 31
twsCFD (twsEquity), 32
twsComboLeg (.twsIncomingMSG), 5
twsConnect, 17, 20, 22, 24, 27, 29
twsConnect2 (twsConnect), 27
twsConnectionTime, 28
twsContract, 5, 13, 17, 20, 22, 24, 29, 32, 33,
35, 36
twsCurrency, 31
twsDEBUG (.twsIncomingMSG), 5
twsDisconnect (twsConnect), 27
twsEquity, 32
twsExecutionFilter (.twsIncomingMSG), 5
twsFOP (twsFuture), 34
twsFUT (twsFuture), 34
twsFuture, 34
twsFutureOpt (twsFuture), 34
twsIND (twsContract), 29
twsIndex (twsContract), 29
twsOPT (twsOption), 35
twsOption, 35
twsOrder, 5, 37
twsPortfolioValue (reqAccountUpdates), 11
twsScannerSubscription, 41