Package 'rib'

October 17, 2024

Title An Implementation of 'Interactive Brokers' API

Version 0.23.1

Description Allows interaction with 'Interactive Brokers' 'Trader Workstation' <https://interactivebrokers.github.io/tws-api/>.
Handles the connection over the network and the exchange of messages.
Data is encoded and decoded between user and wire formats.
Data structures and functionality closely mirror the official implementations.

Depends R (>= 3.4)

Imports R6 (>= 2.4)

License GPL-3

Encoding UTF-8

URL https://github.com/lbilli/rib

BugReports https://github.com/lbilli/rib/issues

NeedsCompilation no

Author Luca Billi [aut, cre]

Maintainer Luca Billi <noreply.section+dev@gmail.com>

Repository CRAN

Date/Publication 2024-10-17 11:50:06 UTC

Contents

enums *Enumerated Types*

Description

Enumerated types are used in few places across the API. These are types that can have only a limited set of named constant values.

These functions facilitate the conversion between integer value and string representation.

Usage

```
map_enum2int(enum, name)

map_int2enum(enum, value)
```

Arguments

enum	name of the enumeration type: *e.g.* "Condition", "FaDataType", "MarketData", "PriceTrigger".
name	string representation of value.
value	integer representation of name.

Value

map_enum2int returns the corresponding value.

map_int2enum returns the corresponding name.

Examples

```
map_enum2int("MarketData", "DELAYED")   # -> 3

map_int2enum("MarketData", 3)           # -> "DELAYED"
```

factory *Helpers*

Description

Helper functions that simplify the customization of common data structures.

Usage

```
IBContract(...)

IBOrder(...)

fCondition(type)
```

Arguments

...	Any combination of named arguments whose names are valid for `Contract` or `Order` respectively.
type	Type of condition: one of `"Price"`, `"Time"`, `"Margin"`, `"Execution"`, `"Volume"` or `"PercentChange"`.

Details

The same result is achieved by making a copy of the respective structures and explicitly reassigning values to the desired fields. The two approaches can be complementary.

Value

`IBContract` returns a `Contract`.

`IBOrder` returns an `Order`.

`fCondition` returns a `Condition`.

See Also

`Contract`, `Order`.

Examples

```
stock <- IBContract(symbol="GOOG", secType="STK", exchange="SMART", currency="USD")

# Equivalent to
stock <- Contract
stock$symbol   <- "GOOG"
stock$secType  <- "STK"
stock$exchange <- "SMART"
stock$currency <- "USD"

order <- IBOrder(action="BUY", totalQuantity=10, orderType="LMT", lmtPrice=99)

condition <- fCondition("Time")
condition$is_more <- TRUE
condition$value   <- "20221114-12:00"
```

IBClient *Client Connection Class*

Description

This is the main class that manages the connection with the 'Trader Workstation', sends requests and handles responses.

Methods

- `IBClient$new()`: creates a new instance.
- `$connect(host="localhost", port, clientId, connectOptions="")`: connects to `host:port` and performs the initial handshake using client identifier `clientId` and additional options `connectOptions`.
- `$checkMsg(wrap, timeout=0.2)`: waits for and process server messages. When available, messages are decoded and handed over to the appropriate callback defined in `wrap`, which must be an instance of a child of `IBWrap`. If `wrap` is missing, messages are read and immediately discarded. Returns the number of messages processed.

  This methods **blocks** up to `timeout` seconds. **Needs to be called regularly**.
- `$disconnect()`: terminates the connection.

This class is modeled after the class `EClient` from the official IB API implementations. In addition to the methods shown above, several others exist that are used to send requests to the server.

Refer to the official documentation for a comprehensive list of the possible requests, including their signatures and descriptions.

See Also

`IBWrap`.

`EClient` definition from the official documentation.

Examples

```
## Not run:
# Instantiate a wrapper
wrap <- IBWrapSimple$new()

# Create a client and connect to a server
ic <- IBClient$new()
ic$connect(port=4002, clientId=1)

# Make a request
stock <- IBContract(symbol="GOOG", secType="STK", exchange="SMART", currency="USD")
ic$reqContractDetails(11, stock)

# Process responses
ic$checkMsg(wrap)

# Disconnect
ic$disconnect()

## End(Not run)
```

IBWrap *Callbacks Wrapper Class*

Description

As the communication with the server is asynchronous, the way to control how inbound messages are processed is via callback functions. The class IBWrap is merely a container for these functions.

Being a base class, its methods are just stubs whose only action is to raise a warning when called.

Customized functionality is provided by defining a child class of IBWrap and overriding the appropriate methods to perform the desired tasks.

These methods are never called directly by the user program, rather they are implicitly invoked within IBClient$checkMsg() when it processes the server responses.

Details

IBWrap is modeled after EWrapper from the official IB API implementations.

The official documentation provides a comprehensive list and description of the available methods, their signatures and usage.

The customization process follows this template:

```
# Class derivation:
IBWrapSimple <- R6::R6Class("IBWrapSimple",
  class=      FALSE,
  cloneable=  FALSE,
  lock_class= TRUE,

  inherit= IBWrap,

  public= list(

    # Customized methods:
   error=         function(id, errorCode, errorString, advancedOrderRejectJson){

                        # Code to handle error messages
                      },

    nextValidId=      function(orderId) {

                        # Code to handle the next order ID
                      },

    contractDetails=  function(reqId, contractDetails) {

                        # Code to handle Contract description
                      },
```

```
    # etc.
  )
)

# Class instantiation:
wrap <- IBWrapSimple$new()

# Use when processing server messages by a client:
ic <- IBClient$new()

ic$checkMsg(wrap)
```

See Also

IBClient.

EWrapper definition from the official documentation.

---

structs *Data Structures*

Description

The data structures used by the API are implemented as R named lists, possibly nested. Templates filled with default values are defined within the package. In order to instantiate them, no elaborate contructor is required but a simple copy will do.

Still, helper functions are available for `Contract` and `Order`.

Usage

```
ComboLeg

Contract

DeltaNeutralContract

ExecutionFilter

Order

OrderCancel

ScannerSubscription

SoftDollarTier

WshEventData
```

See Also

IBContract, IBOrder.

Examples

```
stock <- Contract

stock$symbol   <- "GOOG"
stock$secType  <- "STK"
stock$exchange <- "SMART"
stock$currency <- "USD"
```

Index