IBrokers - Interactive Brokers and R

Jeffrey A. Ryan

December 6, 2010

Contents

Abstract

The statistical language `R` offers a great environment for rapid trade idea development and testing. Interactive Broker's *Trader Workstation* offers a robust platform for execution of these ideas. Previously it was required to use an external language to interface the impressive API capabilities of the *Trader Workstation* — be it Java, Python, C++, or a myriad of other language interfaces, both officially supported or otherwise. What had been lacking was a native `R` interface to access this impressive API. This is now available in the new IBrokers package.

1 Introduction

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.

Interactive Brokers [1] is an international brokerage firm specializing in electronic execution in products ranging from equities to bonds, options to futures, as well as FX, all from a single account. To facilitate this they offer access

to their *Trader Workstation* platform (TWS) through a variety of proprietary APIs. The workstation application is written in Java and all communication is handled via sockets between the client and the TWS.

Access to the API has official support in Java (all platforms), as well as C++, DDE for Excel, and ActiveX (available only for Windows). There are numerous third-party applications and API interfaces available to access the API. These include IbPy - a python translation of the official Java code, a Perl version, and a portable C version.

All of these methods, while useful outside of `R` [3] can't offer true R-level access, including native callbacks and error management, to the TWS API. For this an R solution was required. What this also means is that this implementation is restricted at present to single-threaded API access.

This introduction is broken into two parts. First it will provide an overview of the overall API from Interactive Brokers, as well as examine some of the more common documented methods. The second section will examine the specific implementation of this API in the IBrokers [4] package.

2 The API

The most up to date documentation on the overall API can be found on Interactive Brokers own site [2]. While it is a constantly evolving library - most of the core functionality persists from one version to the next. The principal purpose of the API is to offer a programmatic alternative to manual screen-based trading through Interactive Brokers.

2.1 Data

In order for trade decisions to be automated, data must be processed by the client application. To retrieve real-time data from the TWS there are three primary access methods - `reqMktData`, `reqMktDepth`, and `reqRealTimeBars`. Additionally, limited historical data can be retrieved for many products via the `reqHistoricalData` accessor function. All of these operate with callback handlers within the public API - so it is possible to allow for custom actions based on the incoming data.

New in the beta version of the API are tools to access fundamental data from Thompson Financial. These are not in production versions as of this writing.

2.2 Execution

The API also allows for order execution to be programmatically driven. Through a variety of functions, it is possible to view, modify, and submit orders to be executed by the TWS.

2.3 Miscellaneous functionality

Additional functionality offered includes access to account information, contract details, connection status and news bulletins from the TWS.

3 IB and R

As R offers an ever-growing tooklit of statistical as well as financial functionality, it is becoming a platform of choice for quantitative research and even trading. For institutional users many tools exists to help tie data from external sources into R. Probably the most common is the use of Bloomberg data in R via the RBloomberg package. While many professionals have access to a Bloomberg terminal, it is not usually practical or necessary for smaller or single product traders.

Interactive Brokers gives these users access to many professional features - as well as industry leading executions and prices - all from a solid GUI based application.

To make the transition to programmatically interacting with this application, it had been necessary to use one of the supported or contributed API libraries available. For many users this poses no issue — as many are either unaware of R as a quantitative platform, or make use of it in a limited manner.

For those that use R more frequently it is important to find a workable solution accessible from within R. This is the purpose of IBrokers.

3.1 Getting started

The first step in interacting with the TWS is to create a connection. To do so, it is first necessary to enable the TWS to allow for incoming socket connections. The TWS user manual is the most up to date reference on how to accomplish this, but as of this document it is simply a matter of **Configure > API > Enable ActiveX and Sockets**. You should also add your machine (127.0.0.1) to the **Trusted IP Addresses**.

An new option for connecting to Interactive Brokers API is now available using the IBGateway client provided by Interactive Brokers. This is a low-resource, non-GUI application that can be used in place of the TWS. The only variation to the connection procedure is that the default port for the IBGateway application is 4001, which must be changed in the `twsConnect` call if used. For the rest of the document, TWS will be used but keep in mind all functionality is now available when using the IBGateway as well.

From the R prompt, load the package and establish a connection. Once connected you can then check some basic connection information. Multiple TWS connections are allowed per client, though a distinct clientId is required, and given R's single threaded nature it is not of much value in a single session. Multiple R sessions can run against a single TWS instance though, and in that case would require distinct clientId values. If no further queries are required, `twsDisconnect` will disconnect the R session from the TWS.

```
> library(IBrokers)
> tws <- twsConnect()
> tws
> reqCurrentTime(tws)
> serverVersion(tws)
> twsDisconnect(tws)
```

3.2 Getting data from the TWS

The current IBrokers implementation is focused on retrieving data from the TWS. To that end, five important functions are made available in the API:

reqContractDetails: retrieves detailed product information.

reqMktData: retrieves real-time market data.

reqMktDepth: retrieves real-time order book data.

reqRealTimeBars: retrieves real-time OHLC data.

reqHistoricalData: retrieves historical data.

In addition, due to the complexity of requests, helper functions unique to IBrokers are included to make constructing these requests faster and less error-prone. A family of contract specifier functions used for the above calls includes:

twsContract: create a general Contract object.

twsEquity/twsSTK: wrapper to create equity Contract objects

twsOption/twsOPT: wrapper to create option Contract objects.

twsFuture/twsFUT: wrapper to create futures Contract objects.

twsFuture/twsFOP: wrapper to create futures options Contract objects.

twsCurrency/twsCASH: wrapper to create currency Contract objects.

twsBAG: wrapper to create a combo.

Using the above functions to request market data from the TWS is quite straightforward.

```
> tws <- twsConnect()
> twsFuture("YM", "ECBOT", "200809")
> reqContractDetails(tws, twsEquity("QQQQ"))

> reqMktData(tws, twsEquity("QQQQ"))
```

Unique again to the IBrokers implementation is the use of *passed* callback handlers to the data functions. These are available to help customize output and to facilitate the creation of automated trading programs based on user-defined criteria — coded entirely in R.

Each function has a file argument which allows for persistent data capture to a file. In the case of the real-time methods, file is passed internally to `cat`, which mean that the argument may contain a file name or any valid connection object. Using the default arguments, with some minor allowable exceptions, it is possible to record incoming data to a file for later playback using the same request functions used on live data. The `playback` argument allows for one to control the speed of the returned data. By default `playback=1` the data is retrieved with respect to the original timestamps. This value is multiplied by the difference in time between original message timestamps, increasing the value will slow the playback and decreasing toward 0 will speed up the playback.

```
> reqMktData(tws, twsEquity("SBUX"), CALLBACK = NULL, file = "SBUX.dat")
> twsp <- twsConnect(filename = "SBUX.dat")
```

To playback the saved data, call `reqMktData` again:

```
> reqMktData(twsp)
> reqMktData(twsp, playback = 0)
```

Each data function has special callback methods associated with the expected results. These are documented in the standard R help files of IBrokers. Each distinct message recieved from the TWS invokes a callback function - by default the callbacks built into IBrokers. Users may set these callbacks to functions of there own design, or set them to NULL to return the raw message data, as was show previously when saving data for later playback.

Additionally, the real-time data methods `reqMktData`, `reqMktDepth`, and `reqRealTimeBars` all have a special CALLBACK argument to allow for custom raw message handling. This allows for simplified management of data calls, while offering highly customizable receiver handling. Using this will bypass the built in error management of TWS errors, so the developer should be aware of low-level API interactions before using this feature.

Using CALLBACK involves some understanding of the underlying API process within IBrokers. In general it is as simple as sending a special series of *nul* seperated messages to the TWS, and then interpreting the messages returned. All are character strings, and each request or returned value starts with an identifying code indicating the nature of the content to follow.

The current data methods, based on the official API, simply request the data, and then enter into a loop awaiting the returned messages — branching to message specific callbacks as appropriate.

Utilizing a custom CALLBACK method allows for this loop to be contained entirely within your own code. Setting CALLBACK=NULL will send to raw message level data to `cat`, which in turn will use the `file` argument to that function to either return the data to the standard output, or redirected via an open connection, a file, or a pipe. The details of this can be seen in the help pages for **R**'s `cat`.

Setting CALLBACK=NA will result in a request to the API for the data in question, but will not enter into a loop to manage the incoming messages.

This can be used to launch multiple requests within one **R** session, and then followed with code to read the resulting messages. This is simply a short-cut to hand-coding the requests, and provides no built in message handling logic. This also means that the incoming messages will continue to accumulate until read or cancelled by your code, or a disconnect from the TWS. Obviously this requires the most work, but can offer the most flexibility when it comes to producing a true real-time analysis tool.

An additional caveat with respect to multiple requests within a single session. It is important to recognize that each request must have a unique `tickerId` associated with it, to identify the returned content properly. At present, this must be kept track of by the user.

3.3 Future API access

Future additions will focus on implementing a robust order management collection of functions to allow for a complete data–processing–trade workflow. Additionally, more of the standard API tools will contiue to be integrated into the package.

4 Conclusion

The TWS platform offers the quantitative trader unparalleled access to automated trading tools via its API. IBrokers allows the `R` developer direct access to this API for rapid trade idea development, testing, and eventual execution.

References

[1] Interactive Brokers: *Interactive Broker's* Trader Workstation *and API*, URL http://www.interactivebrokers.com

[2] Interactive Brokers Application Programming Interface: *Interactive Broker's* Trader Workstation *API*, URL http://individuals.interactivebrokers.com/en/p.php?f=programInterface&ib_entity=llc

[3] R Development Core Team: *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org

[4] Jeffrey A. Ryan: *IBrokers: R API to Interactive Brokers Trader Workstation*, R package version 0.3-0, 2008