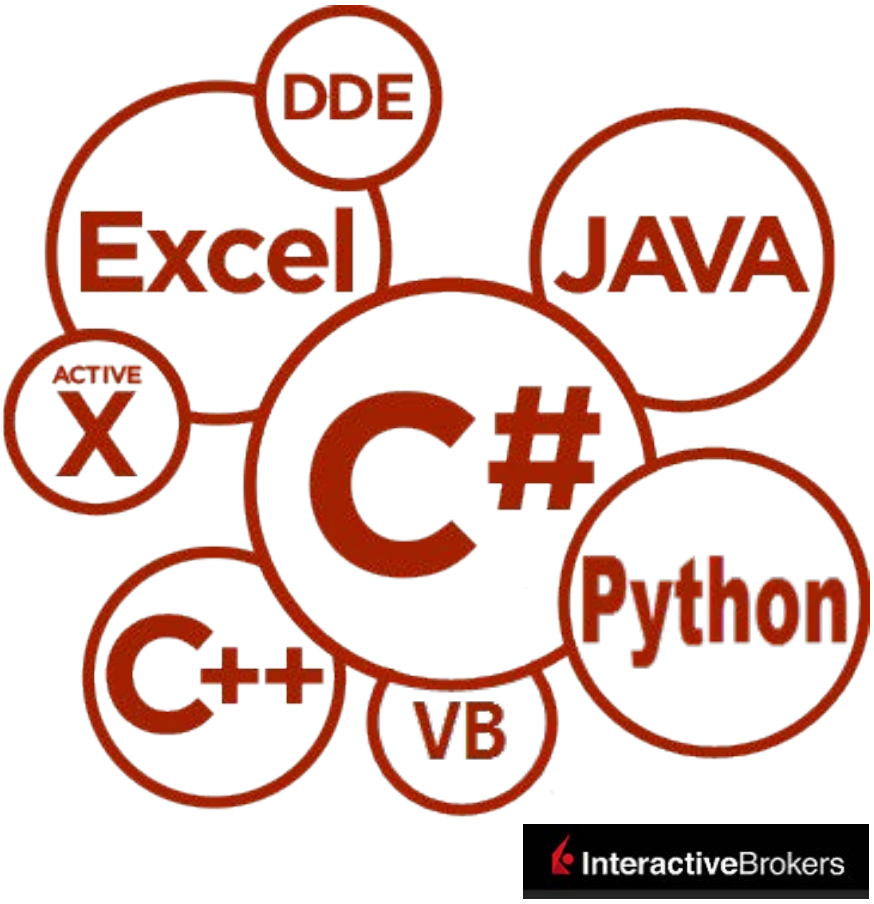
InteractiveBrokers

Trader Workstation API
Reference Guide
Version 973.07

Interactive Brokers
Trader Workstation API
v973.07



This guide was generated by Doxygen from the reference guide:

http://interactivebrokers.github.io/tws-api/

Please refer to the online guide for the most up-to-date information.

Table of Contents

Chapter 1

Introduction

http://interactivebrokers.github.io/tws-api/introduction.html

Code samples in this guide are presented in the order:

- **C#**
- **Java**
- **VB.NET**
- **C++**
- **Python**

The TWS API is a simple yet powerful interface through which IB clients can automate their trading strategies, request market data and monitor your account balance and portfolio in real time.

2.1 Audience

Our TWS API components are aimed at experienced professional developers willing to enhance the current TWS functionality. Regrettably, Interactive Brokers cannot offer any programming consulting. Before contacting our API support, please always refer to our available documentation, sample applications and `Recorded Webinars`

2.2 How to use this guide

This guide reflects the very latest version of the TWS API -**9.72 and higher**- and constantly references the Java, VB, C#, C++ and Python **Testbed** sample projects to demonstrate the TWS API functionality. All code snippets are extracted from these projects and we suggest all those users new to the TWS API to get familiar with them in order to quickly understand the fundamentals of our programming interface. The **Testbed** sample projects can be found within the **samples** folder of the TWS API's installation directory.

2.3 Requirements

- • The TWS API is an interface to TWS or IB Gateway, and as such requires network connectivity to a running instance of one of these programs.
  • To obtain the TWS API source and sample code, download the `API Components`.
  • To make use of TWS API 9.72+, will require `TWS build` 952.x or higher.
  • A working knowledge of the programming language our **Testbed** sample projects are developed in.

- • To obtain the TWS API source and sample code, download the `API Components`.

  • To make use of TWS API 9.72+, will require `TWS build` 952.x or higher.

  • A working knowledge of the programming language our **Testbed** sample projects are developed in.

  • Java 8 or higher is required for running the Java API client.

- • To obtain the TWS API source and sample code, download the `API Components`.
  - To make use of TWS API 9.72+, will require `TWS build` 952.x or higher.
  - A working knowledge of the programming language our **Testbed** sample projects are developed in.
  - Microsoft .Net Framework 4.5 or higher is required for running the VB API client.

- • To obtain the TWS API source and sample code, download the `API Components`.
  - To make use of TWS API 9.72+, will require `TWS build` 952.x or higher.
  - A working knowledge of the programming language our **Testbed** sample projects are developed in.
  - A compiler that supports C++11 is required for running the C++ API client.
  - For Windows users who use Visual Studio, a version of 2012 or higher is required.

- • To obtain the TWS API source and sample code, download the `API Components` (API version 9.73 or higher is required).
  - To make use of TWS API 9.73+, will require `TWS build` 952.x or higher.
  - A working knowledge of the programming language our **Testbed** sample projects are developed in.
  - Python version 3.1 or higher is required to interpret Python API client.

2.4 Limitations

Our programming interface is designed to automate some of the operations a user normally performs manually within the TWS Software such as placing orders, monitoring your account balance and positions, viewing an instrument's live data... etc. There is no logic within the API other than to ensure the integrity of the exchanged messages. Most validations and checks occur in the backend of TWS and our servers. Because of this it is highly convenient to familiarize with the TWS itself, in order to gain a better understanding on how our platform works. Before spending precious development time troubleshooting on the API side, it is recommended to first experiment with the TWS directly.

Remember: If a certain feature or operation is not available in the TWS, it will not be available on the API side either!

2.4.1 Requests

The TWS is designed to accept up to **fifty** messages per second coming from the **client** side. Anything coming from the client application to the TWS counts as a message (i.e. requesting data, placing orders, requesting your portfolio... etc.). This limitation is applied to **all** connected clients in the sense were all connected client applications to the same instance of TWS combined cannot exceed this number. On the other hand, there are **no limits** on the amount of messages the TWS can send to the client application.

2.4.2 Paper Trading

If your regular trading account has been approved and funded, you can use your Account Management page to open a `Paper Trading Account` which lets you use the full range of trading facilities in a simulated environment using real market conditions. Using a Paper Trading Account will allow you not only to get familiar with the TWS API but also to test your trading strategies without risking your capital. Note the paper trading environment has inherent `limitations`.

Chapter 2

Initial Setup

http://interactivebrokers.github.io/tws-api/initial_setup.html

The TWS API is an interface to IB's standalone trading applications, TWS and IB Gateway. These are both standalone, Java-based trading applications which were designed to require the use of a graphical user interface for the security of user authentication. For that reason, "headless" operation is not supported.

3.1 The Trader Workstation

Our market maker-designed IB Trader Workstation (TWS) lets traders, investors, and institutions trade stocks, options, futures, forex, bonds and funds on over 100 markets worldwide from a single account. The TWS API is a programming interface to TWS and as such, it forcefully requires a TWS to connect to. To use version 9.72+ of the API, it is necessary to have TWS version 952 or higher.

3.2 The IB Gateway

As an alternative to TWS for API users, IB also offers IB Gateway (IBG). From the perspective of an API application, IB Gateway and TWS are essentially identical; both represent a server to which an API client application can open a socket connection once the user has first authenticated a session. In either case (TWS or IBG), the user must authenticate a session by manually entering credentials into the login window- for security reasons, a headless session of TWS or IBG without a GUI is not supported. From the user's perspective, IB Gateway is a lighter application which consumes about 40% fewer resources and has some advantages over using an API connection to TWS- most noticeably, it does not have the designed limitation that an autologoff time is programmed when it must shutdown to be restarted. IB Gateway does not update automatically and it is recommended to upgrade to a current version from the website every few months or less.

- *There are however some times in which it will also be necessary to restart IB Gateway*, for instance if there is a change to IB's contract database because of a ticker symbol changed or the introduction of a new contract, it will be necessary to restart IBG to access the updated contract information.

The advantages of TWS over IBG is that it provides the end user with many tools (Risk Navigator, OptionTrader, BookTrader, etc) and a graphical user interface which can be used to monitor an account or place orders. For beginning API users, it is recommended to first become acquainted with TWS before using IBG.

For simplicity, this guide will mostly refer to the TWS although the reader should understand that for the TWS API's purposes, TWS and IB Gateway are synonymous.

3.3 Enable API connections

Before any client application can connect to the Trader Workstation, the TWS needs to be configured to listen for incoming API connections on a very specific port. By default when TWS is first installed it will not allow API connections. IBG by contrast accepts socket-based API connections by default. To enable API access in TWS, navigate to the TWS' API settings at Edit -> Global Configuration -> API -> Settings and make sure the "Enable ActiveX and Socket Clients" option is activated as shown below:

Also important to mention is the "Socket port". By default a production account TWS session will be set for socket port 7496, and a paper account session will listen on socket port 7497. However these are just default values chosen because they are almost always available on any computer. They can be changed to any open socket port, as long as the socket ports specified in the API client and TWS settings match. If there are multiple TWS sessions on one computer, the socket port is used to distinguish the TWS session. Since only one application can listen on one port at a time you will need to assign different ports to each running TWS.

Important: when running paper and live TWS on the same computer, make sure your client application is connecting to the right TWS!

3.4 Read Only API

The API Settings dialogue allows you to configure TWS to note accept API orders with the "Read Only" setting. By default, "Read Only" is enabled as an additional precautionary measure. Information about orders is not available to the API when read-only mode is enabled.

3.5 Master Client ID

By default the "Master Client ID" field is unset. To specify that a certain client should *automatically* receive updates about all open orders, as well as commission reports from orders placed from all clients, the client's ID should be set as the Master Client ID in TWS or IBG Global Configuration. The clientID is specified from an API client application in the initial function call to **IBApi::EClientSocket::eConnect** (p. **??**).

3.6 Installing the API source

The API itself can be downloaded and installed from:

`http://interactivebrokers.github.io/`

Many third party applications already have their own version of the API which is installed in the process of installing the third party application. If using a third party product, it should first be verified if the API must be separately installed and what version of the API is needed- many third party products are only compatible with a specific API version.

The Windows version of the API installer will create a directory "C:\\TWS API\" for the API source code in addition to automatically copying two files into the Windows directory for the ActiveX/DDE and C++ APIs. ***It is important that the API installs to the C: drive***, as otherwise API applications may not be able to find the associated files. The Windows installer also copies compiled dynamic linked libraries (DLL) of the 32 versions of the ActiveX control TWSLib.dll, C# API CSharpAPI.dll, and C++ API TwsSocketClient.dll. Starting in version 973.03, the Windows installer also installs a 32 bit version of the RTDServer control. To use a 64 bit application which loads the API as a dynamic library, it is necessary to compile and install a 64 bit version of the desired control.

3.7 Changing the installed API version

(On Windows Only)

If a different version of the ActiveX (v9.71 or lower) or C++ API is required than the one currently installed on the system, there are additional steps required to uninstall the previous API version to manually remove a file called "TwsSocketClient.dll":

1) Uninstall the API from the "Add/Remove Tool" in the Windows Control Panel as usual

2) Delete the C:\TWS API\ folder if any files are still remaining to prevent a version mismatch.

3) Locate the file "C:\Windows\SysWOW64\TwsSocketClient.dll". Delete this file.

4) Restart the computer before installing a different API version.

Chapter 3

Using Third Party API Platforms

http://interactivebrokers.github.io/tws-api/third_party.html

Third party software vendors make use of the TWS' programming interface (API) to integrate their platforms with Interactive Broker's. Thanks to the TWS API, well known platforms such as Ninja Trader or Multicharts can interact with the TWS to fetch market data, place orders and/or manage account and portfolio information.

- **It is important to keep in mind that most third party API platforms are not compatible with all IB account structures**. Always check first with the software vendor before opening a specific account type or converting an IB account type. For instance, many third party API platforms such as NinjaTrader and Trade↩ Navigator are **not** compatible with IB linked account structures, so it is highly recommended to first check with the third party vendor before linking your IB accounts.

A non-exhaustive list of third party platforms implementing our interface can be found in our `Investor's Marketplace`. As stated in the marketplace, the vendors' list is in no way a recommendation from Interactive Brokers. If you are interested in a given platform that is not listed, please contact the platform's vendor directly for further information.

4.1 Frequently Asked Questions

Below is a list of frequently asked questions. All answers are given considering a standard usage of the TWS. Note that some vendors might provide an additional customisation level to simplify things. If the below description does not reflect the way you operate your third party software with the TWS, please contact your vendor directly.

- **How to connect a 3rd party platform to Interactive Brokers' Trader Workstation**
- **Where to get support for a third party software connecting to the TWS.**
- **My program's vendor did not find any issue on its side and asked me to contact Interactive Brokers directly.**
- **TWS generates warning messages that block my orders being automatically transmitted**
- **I cannot see any market data in my third party program**
- **I do have the Live Data Subscriptions I need but when using my paper trading user name I am still unable to obtain it.**
- **I am obtaining a message saying "Historical data request pacing violation"**
- **I am obtaining a "HMDS query returned no data" message.**

- **I cannot chart CFDs from my third party program yet the TWS shows the data correctly.**
- **Can I connect simultaneously to my live and paper TWS?**
- **The charts shown by my charting software differ from the ones shown by the TWS**
- **Is there a fee involved to receive data from the API?**
- **Can the TWS API be used with trial accounts?**
- **What types of APIs are available?**
- **What are differences between connecting to the TWS API and using a FIX/CTCI connection?**
- **Can I use another trading application (IBKR mobile, WebTrader, TWS) while an API program is con-nected?**
- **Can the TWS API be used with any IB account?**
- **Why does an order from the API appear as untransmitted in TWS with a 'T' button next to it?**
- **Is autolaunching of TWS or IB Gateway supported by IB?**
- **Is it possible to disable the daily auto-logoff requirement of TWS?**
- **Do you recommend any third party products or programming consultants?**
- **What are the differences between using an API application with TWS and IB Gateway?**
- **Is it possible to run TWS or IB Gateway on a headless server?**
- **Do I ever need to upgrade the TWS API?**
- **The data indicator in IB Gateway is red. Is something wrong?**
- **How do I report a problem to the API Support team?**
- **Are historical account positions or account values available from the TWS API?**
- **Does IB provide hosting services for custom algorithms?**
- **Is there an Excel API for MacOS?**

4.1.1 How to connect a 3rd party platform to Interactive Brokers' Trader Workstation

Connecting any third party program to the TWS requires you to enable API connectivity on the TWS itself as explained in **Enable API connections** section. Your third party program will need to provide means for you to specify at least a **host** and a **port**. In the vast majority of cases the third party program will be running on the exact same computer as the TWS therefore the host IP can be specified as **127.0.0.1** whereas the port needs to be the exact same as the one configured in the TWS' API Settings, typically **7496** or **7497**. Below is an illustration showing one of our API sample applications highlighting the typical connectivity fields a program connecting to the TWS should provide. Note there is an additional field for "clientId". You can set this id to any positive integer including zero or to whatever your third party application's provider recommends.

Note: for vendor-specific instructions please contact your third party provider directly.

4.1.2 Where to get support for a third party software connecting to the TWS.

Interactive Brokers cannot provide any kind of support or advice for software not developed by IB itself. Depending on the nature of your inquiry, our support staff's advice might be quite limited since in most of cases it is not realistic to reverse-engineer a third party program in order to understand how is it using our API. Because of this, the third party vendor's support team should always be the first contact option.

4.1.3 My program's vendor did not find any issue on its side and asked me to contact Interactive Brokers directly.

There will be occasions when a given operation will not be fulfilled as expected not because of a malfunction in either platform but because of the business logic involved. The typical behaviour of the TWS is to either perform the requested operation or to return an explanatory message which will point you in the right direction. It is the duty of the third party program to clearly show these TWS' messages within its own user interface. Without a relevant error message our support team will not be able to give any advice.

In case of a malfunction on Interactive Broker's side, clear and concise technical information needs to be provided including evidence of the malfunction in the form of TWS or API message log files as detailed in **Log Files** section of this guide. Collecting all the needed information is not a trivial process and might require a very detailed knowledge of our API. To prevent being caught in between support teams, please request your third party vendor to contact us directly via email to `api@interactivebrokers.com` with as much information as possible.

4.1.4 TWS generates warning messages that block my orders being automatically transmitted

There are precautionary settings in TWS that are designed to be a order safety check. TWS would usually generate a pop-up warning dialogue, or sent back an error message via the API, when there is any violation to the pre-define precautionary settings in `TWS Presets`.

For users who uses a third party software to place orders but also receives data feed from a different vendor other than IB, TWS would also generate a default warning *"You are trying to submit an order without having market data for this instrument"* on receiving orders from third party. The checked order will not be transmitted automatically unless user click "Transmit" button of the order in TWS.

TWS precautions can be bypassed by navigating to *File/Edit* $\rightarrow$ *Global Configuration* $\rightarrow$ *API* $\rightarrow$ *Precautions*, and check the box *"Bypass Order Precautions for API Orders"*. Once this is done, API orders placed from a third party software will not be checked by TWS precautions.

4.1.5 I cannot see any market data in my third party program

As explained in our **Streaming Market Data** page, in order to be able to pull market data from the TWS, you need to acquire the **Live Market Data** of the product(s) you are interested in.

4.1.6 I do have the Live Data Subscriptions I need but when using my paper trading user name I am still unable to obtain it.

Make sure you are **Sharing Market Data Subscriptions**

4.1.7 I am obtaining a message saying "Historical data request pacing violation"

Please refer to our **Historical Data Limitations** page

4.1.8 My third party program shows "No data of type EODChart is available" when trying to load a chart

This message is returned when the requested End of Day (EOD) market data is not available in our systems. You can easily verify this by loading the exact same chart using the TWS to obtain the same result:

Please contact our General Support team's Market Data department for further information.

4.1.9 I am obtaining a "HMDS query returned no data" message.

Sometimes the data prior to the specified requested date is also not available for several reasons. Suppose a product started quoting (generating data) on 1st January 2016 but your third party program requests data prior to this date. To prevent this error message adjust your third party program's charting parameters accordingly.

4.1.10 I cannot chart CFDs from my third party program yet the TWS shows the data correctly.

Except for **Index CFDs**, CFDs do not have any market data of their own. What the TWS displays is the CFD's **underlying contract's** data. Whenever you try to fetch non-Index CFD data from the TWS, you will obtain an error message asking you to pull it's underlying contract's data instead. Some third party programs' user interface only allow placing orders via their charts. Given that no data can be loaded for these products and the impossibility of the third party program to apply a similar conversion as the TWS, you might find your third party software not able to place orders as a consequence!

4.1.11 How can I connect my third party program to my paper trading account?

Connections via the TWS API are not aware of the user name with which you are logged in with in your TWS. From this point of view the API makes no difference between live or paper trading. Since third party programs only connect any running TWS on the specified host and port it only takes you to launch the TWS and log into it with your paper trading credentials, make sure you **Enable API connections** and connect to it from your third party application using the same procedure.

4.1.12 Can I connect simultaneously to my live and paper TWS?

From our side, yes. You can launch as many instances of the TWS as you need using different user name/password combinations. It is crucial though to make sure each TWS is listening on a different **port** as described in the **Enable API connections** section. Note that your might as well need to launch multiple instances of your third party program and/or have a way of telling when is your program using the paper or the live accounts.

4.1.13 The charts shown by my charting software differ from the ones shown by the TWS

Given that the historical data sent down the TWS API comes from the same source as the one displayed by the TWS itself it is almost impossible for it to differ.

Some charting platforms circumvent our **Historical Data Limitations** by combining real time and historical data. Since our real time market data is **not** tick-by-tick, bars built from it will hardly match those retrieved from our historical market data service.

Alternatively, you might as well be comparing different charts without noticing. A chart displaying data from NYSE will never be the same as another built from the ARCA exchange or from our SMART routing strategy. inadvertently you might as well be looking at TRADES on one side while having MIDPOINT or BID_ASK on another. Another common mistake involves having different timezones between the TWS and your client application.

4.1.14 Is there a fee involved to receive data from the API?

Streaming real time data or receiving historical bars from the API requires streaming level 1 market data subscriptions. Subscriptions for instruments other than forex, bonds, and index CFDs incur a monthly fee.
`http://interactivebrokers.github.io/tws-api/market_data.html`

4.1.15 Can the TWS API be used with trial accounts?

Yes it is possible to connect an API application to a trial account. However it is not possible to receive real time market data or historical candlesticks for most instruments from the TWS API with a trial account login.

4.1.16 What types of APIs are available?

There is an API for Traders Workstation (TWS API). It is the most full-featured and can be used by all clients with trading access. There is a WT Web API for white-branded advisors and introducing brokers that provides streaming and historical data. There is a REST WebAPI for third party companies and institutions. It can be used to place stock and forex orders, receive market data snapshots, and receive account and portfolio information. https://www.interactivebrokers.com/en/index.php?f=1325

4.1.17 What are differences between connecting to the TWS API and using a FIX/CTCI connection?

The TWS API is an interface to TWS or IB Gateway. It provides many functionalities, such as the ability to receive market data, place orders, and receive account information. The TWS API requires that the user first login to either TWS or IB Gateway, both standalone desktop applications. The TWS API has an order rate limitation of 50 orders per second.

FIX/CTCI connectivity can be configured to connect either with IB Gateway or directly to IB. Unlike the TWS API, FIX/CTCI is only for order placement and can not be used to receive market data. Also, FIX/CTCI has monthly minimum commissions involved.

4.1.18 Can I use another trading application (IBKR mobile, WebTrader, TWS) while an API program is connected?

To connect to the same IB account simultaneously with multiple trading applications it is necessary to have multiple usernames. Additional usernames can be created through Account Management free of charge. Market data subscriptions however only apply to individual usernames so the fees would be charged separately.

4.1.19 Can the TWS API be used with any IB account?

Most third party API applications do not support all IB account structures, so it is highly recommended to consult with the third party software vendor before opening or converting to a specific IB account type to use with a third party application.

4.1.20 Why does an order from the API appear as untransmitted in TWS with a 'T' button next to it?

If an order appears in TWS as untransmitted and is not sent to IB, there are generally three causes: (1) There is an error in the order preventing transmission (2) There is a TWS precautionary setting preventing transmission to IB. Precautionary settings can be globally overridden in TWS through the settings in Global Configuration at API -> Precautions -> Bypass Order Precautions for API Orders (3) The order was sent with the 'transmit' boolean flag in the API Order class set to False. By default, its value is True.

4.1.21 Is autolaunching of TWS or IB Gateway supported by IB?

Unfortunately for security reasons auto-launching of TWS or IB Gateway is not supported. Both applications are designed to require the user to manually enter his or her credentials into the UI.

4.1.22 Is it possible to disable the daily auto-logoff requirement of TWS?

It is possible to adjust the time of auto-logoff, but not disable it. However, IB Gateway can be substituted for TWS for use with an API application and it does not have the daily logoff requirement.

4.1.23 Do you recommend any third party products or programming consultants?

IB does not recommend particular third parties, but a list is maintained on the Investors Marketplace. `https://gdcdyn.interactivebrokers.com/Universal/servlet/MarketPlace.MarketPlaceServlet`

4.1.24 What are the differences between using an API application with TWS and IB Gateway?

From the point of view of an API application, TWS and IB Gateway are essentially identical. TWS additionally offers the user the ability to directly view positions, trades, and market data, and provides a number of tools for trading, research, and analysis. IB Gateway has a simple graphical user interface that is only used for modifying settings. The advantages of IB Gateway is that it consumes 40% fewer resources and can run for longer periods without automatically closing down. `http://interactivebrokers.github.io/tws-api/initial_setup.html`

4.1.25 Is it possible to run TWS or IB Gateway on a headless server?

Both TWS and IB Gateway are designed to have a user interface for the client to enter their account credentials. For that reason, headless or GUI-less operation is not supported.

4.1.26 Do I ever need to upgrade the TWS API?

TWS is very backwards compatible with the API so it is not necessary to upgrade the API when upgrading to a new version of TWS. Generally the only reason it is necessary to upgrade the API is to take advantage of a new feature introduced in a more recent API version.

4.1.27 The data indicator in IB Gateway is red. Is something wrong?

It is normal for the market data farm connection indicator to stay red until a market data request is made from the API application. The farms can also turn red after extended inactivity.

4.1.28 How do I report a problem to the API Support team?

Diagnosing specific issues will generally require that API logging is enabled in TWS or IB Gateway when the issue occurs. In TWS this is done by navigating to Global Configuration -> API -> Settings and checking the box "Create API Message Log", and setting the logging level to "Detail" In IB Gateway, these settings are found at Configure -> Settings -> API -> Settings If an issue occurs after logs are enabled, they can be uploaded using the combination Ctrl-Alt-Q and then clicking Submit. Please let API Support know logs have been uploaded.

4.1.29 Are historical account positions or account values available from the TWS API?

Since the API cannot provide information not available in TWS, historical portfolio information is not available. It is available from statements and flex queries in Account Management.

4.1.30 Does IB provide hosting services for custom algorithms?

Unfortunately no, web hosting is not provided.

4.1.31 Is there an Excel API for MacOS?

The Excel APIs require a Windows computer with Microsoft Excel.

Chapter 4

Excel APIs http://interactivebrokers.github.io/tws-api/excel_apis.html

5.1 Available Excel APIs

There are several API technologies available for Microsoft Excel. Since they utilize Windows technologies the Excel APIs require a Windows OS.

- **RTD Server for Excel**
- **Dynamic Data Exchange**
- **ActiveX for Excel API**

Important: Sample spreadsheet applications are distributed with the API download for each of the API technologies (RTD Server, ActiveX, DDE). It is important to keep in mind that the sample applications are intended as simple demonstrations of API functionality for third party programmers. They do not have robust error handling functionality and are not intended to be used as production level trading tools.

Recorded webinars providing an introduction to Excel API technologies are available from the IB website at↩ : `Recorded Excel API Webinars`

5.2 Excel API comparison

	RTDServer	DDE	ActiveX
Full API functionality	Not currently	No	Yes
Easy to use formulas	Yes	Sometimes	No
Use without VBA	Yes	Sometimes	No
Designed to not overwhelm Excel	Yes	No	No
Open Source	Yes	No	Yes
Market Data Refresh rate	250 ms	250 ms	1 sec
Sample compatible with 64 bit Excel	No∗	Yes	Yes∗∗

∗expected in next API release, to be named 973.07

∗∗API ActiveX installer is compatible with both 32 and 64 bit applications starting with v973.05

5.3 Limitations of Microsoft Excel APIs

By design, Microsoft Excel gives precedence to the user interface over the data connection to other applications. For that reason, Excel only receives updates when it is in a 'ready' state, and may ignore data sent for instance when a modal dialogue box is displayed to the user, a cell is being edited, or Excel is busy doing other things. A new Excel Real Time Data server (RTD) API has been introduced to help address some of these limitations, but they are inherent to Excel as a trading application and not specific to an API technology.

5.4 RTD Server for Excel

5.4.1 Introduction

TWS RTD Server API is a dynamic link library which allows user to request real-time market data from TWS via API using Microsoft Excel®. The TWS RTD Server API directly uses the C# API Client source, which connects to TWS via the socket. It allows displaying streaming live (or 15-minute delayed) market data in Excel by entering formulas into an Excel cell following a specific syntax.

Note: At the current stage, only top-level market data is supported via TWS RTD Server API. No trading capability or other data types are supported. Both Delayed and Real-Time data are supported via TWS RTD Server API. **Market Data Subscription** is required for requesting live streaming market data.

5.4.2 What You Will Need

5.4.2.1 - Windows Operating System

Since the TWS RTD Server API technology directly refers to the C# API client source functions, it is supported on `Windows Environment` only.

5.4.2.2 - API version 9.73.03+

You need to download `IB API` Windows version 9.73.03 or higher and install on your computer. Once you have installed the API, you can verify the API Version by checking *C:\TWS API\API_VersionNum.txt* by default.

5.4.2.3 - TWS (or IB Gateway) Build 963+

By default, market data requests sent via TWS RTD Server will automatically request for all possible **Generic Tick Types** (p. **??**). There are several generic tick types being requested that are only supported in TWS 963 or higher. Sending any RTD market data request with default generic tick list to an old build of TWS will trigger a "TwsRtd↩ Server error" indicating incorrect generic tick list is sent. Make sure a `TWS builds 963+` is downloaded from IB website and kept running at the background for TWS RTD Server API to function properly.

5.4.2.4 - Enable Socket Client in TWS (or IB Gateway)

Since the TWS RTD Server API directly refers to the C# API source, RTD market data requests will be sent via the socket layer. Please make sure to **Enable ActiveX and Socket Client** settings in your TWS.

Please also be mindful of the socket port that you configure in your TWS API settings. The default socket port TWS will listen on is **7496** for a live session, and **7497** for a paper session. It is further discussed in section **Connection Parameters** that TWS RTD Server connects to port **7496** by default, and you are able to customize the port number to connect by specifying pre-defined **Connection Parameters** or using rtd_complex_syntax string "port=<port>". You can use any valid port for connection as you wish, and you just need to make sure that the port you are trying to connect to via the API is the same port your TWS is listening on.

5.4.2.5 - Microsoft Excel®

After installing the API, the pre-compiled RTD library file (located at *C:\TWS API\source\csharpclient\TwsRtd↩ Server\bin\Release\TwsRtdServer.dll* by default) registered on your computer will be in 32-bit by default for API versions from 973.03 to 973.06. If you are using 64-bit Microsoft Excel, you would need to re-compile RTD server dll file into 64-bit and register the library by re-building the RTD source solution using Visual Studio. Please refer to the `TWS Excel APIs, featuring the RealTimeData Server` recorded webinar for more information. Beginning in API v973.07∗ it is expected that the API installer for RTD Server will be compatible with both 32 bit and 64 bit Excel (∗ expected version number).

5.4.3 TWS RTD Server Formula Syntax

Customer can request market data by entering the following formula with corresponding parameters into an Excel spreadsheet cell:

`=RTD(`**`ProgID, Server, String1, String2, ...`**`)`

where

- **ProgID** = "Tws.TwsRtdServerCtrl"
- **Server** = "" (empty string)
- **String1**, **String2**, ... is a list of strings representing **Ticker**, **Topic**, **Connection Parameters** or other **Complex Syntax** strings.

Note: *TWS RTD Server API formula is* ***not*** *case-sensitive.*

There are three ways to compose an RTD Formula:

- **Simple Syntax**
- **Complex Syntax**
- **Mixed Syntax**

5.4.4 Syntax Samples

A resourceful **Syntax Samples** page is provided for demonstration of RTD formulas categorized by security type using different syntaxs.

- **Forex Pairs**
- **Stocks**
- **Indexes**
- **CFDs**
- **Futures**
- **Options**
- **Futures Options**
- **Bonds**
- **Mutual Funds**
- **Commodities**
- **Spreads**

Besides reading through the written documentation, you can also watch the `TWS Excel APIs, featuring the RealTimeData Server` recorded webinar for a more interactive tutorial video.

5.4.5 Outgoing message rate limit

- It is important to keep in mind the 50 message/second API limit applies to RTD Server in the same way as other socket-based API technologies. So the Excel spreadsheet can send no more than 50 messages/second to TWS. Each subscription or cancellation request counts as 1 message (messages in the opposite direction are not included). So a spreadsheet can have hundreds of streaming tickers, but the subscriptions must be spread out over time so that no more than 50 new subscriptions are made per second, or the spreadsheet can become disconnected.

5.4.6 Change Data Refresh Rate

- **Change Data Refresh Rate**

5.4.7 Troubleshooting Common Errors

- **Troubleshooting Common Errors**

5.4.8 Simple Syntax

Basic components of a Simple Syntax RTD formula are **ProgID**, **Server**, **Ticker (**p. **??)**, **Topic** and **Connec-tion Parameters** (optional):

`=RTD(ProgID, Server, Ticker, Topic, ConnectionParams...)`

where

- **ProgID** = "Tws.TwsRtdServerCtrl"
- **Server** = "" (empty string)

5.4.8.1 Ticker

Besides **ProgID** and **Server**, the first string should represent the **Ticker** in Simple Syntax.

To define a contract **Ticker** properly, you would need to find the correct contract attributes first. The easiest way to find contract attributes is to directly look at the **Contract Description** page in TWS.

The syntax for **Ticker** should strictly follow the below squence:

Forex Contract:
"CURRENCY1.CURRENCY2/CASH"

e.g. EUR.USD Forex can be defined as "EUR.USD/CASH"

Other Contract Types:

"SYMBOL@EXCHANGE/PRIMEXCH/SECTYPE/EXPIRATION/RIGHT/STRIKE/CURRENCY"

e.g. The E-mini futures can be defined as "ES@GLOBEX//FUT/201712///USD"

Notes:

1. *Not all contract attributes are required to be specified. You can leave the field to be blank to make that field un-specified. Sequencially, if you only need to specify several contract attributes at the begining part of the **Ticker** string, you can leave out the rest of the string entriely as well. For example, instead of specifying "IBM@SMART//////", "IBM@SMART" would be sufficient to define the contract properly.*
2. *There are several default contract attributes in the Ticker string. If you leave them un-specified, they will take the default values as following:*
   - ***EXCHANGE =** "SMART"*
   - ***SECTYPE =** "STK"*
   - ***CURRENCY =** "USD"*

*For example, **Ticker** = "IBM" is the same as "IBM@SMART//STK////USD".*

*See more **Syntax examples** (*p. **??**).

5.4.8.2 Topic

The second (or other strings) can be **Topic**. **Topic** string defines the tick type you would like to receive in the formula cell. **Topic** can be specified within the **Ticker** string, or as a separate string.

For exmaple, the two formulas below both request the Bid Size for IBM from ISLAND exchange, where the first formula includes **Topic** in **Ticker** and the second formula specifies **Topic** as a separate string:

```
=RTD("Tws.TwsRtdServerCtrl",,"IBM@ISLAND BidSize")
```

```
=RTD("Tws.TwsRtdServerCtrl",,"IBM@ISLAND", "BidSize")
```

If no **Topic** string is defined, the **Topic** will be defaulted to *"Last"*.

For example, the below formula will request the last price for IBM from Island exchange:

```
=RTD("Tws.TwsRtdServerCtrl",,"IBM@ISLAND")
```

5.4.8.2.1 Basic Tick Types

The table below shows a full list of available basic tick types that can be specified for the **Topic**:

Tick Name	Topic String	Description
Bid Size	"BidSize"	Number of contracts (or lots) offered at the bid price.
Bid Price	"Bid"	Highest bid price for the contract.
Ask Price	"Ask"	Lowest offer price for the contract.
Ask Size	"AskSize"	Number of contracts (or lots) offered at the ask price.
Last Price	"Last"	Last price at which the contract traded.
Last Size	"LastSize"	Number of contracts or lots traded at the last price.
High	"High"	High price for the day.
Low	"Low"	Low price for the day.
Volume	"Volume"	Trading volume for the day for the selected contract (Volume for US Stocks are quoted in lots. The actual number of shares in volume can be calculated by multiplying 100).
Close Price	"Close"	The last available closing price for the previous day. For US Equities, we use corporate action processing to get the closing price, so the close price is adjusted to reflect forward and reverse splits and cash and stock dividends.
Open Price	"Open"	Today's opening price. The official opening price requires a market data subscription to the native exchange of a contract.
Last Exchange	"LastExch"	The exchange where the Last Price is provided from.
Bid Exchange	"BidExch"	The exchange where the Bid Price is provided from.
Ask Exchange	"AskExch"	The exchange where the Ask Price is provided from.
Last Timestamp	"LastTime"	Time of the last trade (in UNIX time).
Halted	"Halted"	Indicates if a contract is halted. See **Halted**
Bid Implied Volatility	"BidImpliedVol"	Implied volatility calculated from option bid prices.
Bid Delta	"BidDelta"	Delta calculated from the option bid prices.
Bid Option Price	"BidOptPrice"	Current bid price for the option contract.
Bid PV Dividend	"BidPvDividend"	The present value of dividends expected on the option's underlying.
Bid Gamma	"BidGamma"	The option gamma value calculated from the option bid prices.

Tick Name	Topic String	Description
Bid Vega	"BidVega"	The option vega value calculated from the option bid prices.
Bid Theta	"BidTheta"	The option theta value calculated from the option bid prices.
Bid Price of Underlying	"BidUndPrice"	The current bid price of the option underlying.
Ask Implied Volatility	"AskImpliedVol"	Implied volatility calculated from option ask prices.
Ask Delta	"AskDelta"	Delta calculated from the option ask prices.
Ask Option Price	"AskOptPrice"	Current ask price for the option contract.
Ask PV Dividend	"AskPvDividend"	The present value of dividends expected on the option's underlying.
Ask Gamma	"AskGamma"	The option gamma value calculated from the option ask prices.
Ask Vega	"AskVega"	The option vega value calculated from the option ask prices.
Ask Theta	"AskTheta"	The option theta value calculated from the option ask prices.
Ask Price of Underlying	"AskUndPrice"	The current ask price of the option underlying.
Last Implied Volatility	"LastImpliedVol"	Implied volatility calculated from option last prices.
Last Delta	"LastDelta"	Delta calculated from the option last prices.
Last Option Price	"LastOptPrice"	Current last price for the option contract.
Last PV Dividend	"LastPvDividend"	The present value of dividends expected on the option's underlying.
Last Gamma	"LastGamma"	The option gamma value calculated from the option last prices.
Last Vega	"LastVega"	The option vega value calculated from the option last prices.
Last Theta	"LastTheta"	The option theta value calculated from the option last prices.
Last Price of Underlying	"LastUndPrice"	The current last price of the option underlying.
Model Implied Volatility	"ModelImpliedVol"	Implied volatility calculated from option model prices.
Model Delta	"ModelDelta"	Delta calculated from the option model prices.
Model Option Price	"ModelOptPrice"	Current model price for the option contract.
Model PV Dividend	"ModelPvDividend"	The present value of dividends expected on the option's underlying.
Model Gamma	"ModelGamma"	The option gamma value calculated from the option model prices.
Model Vega	"ModelVega"	The option vega value calculated from the option model prices.
Model Theta	"ModelTheta"	The option theta value calculated from the option model prices.
Model Price of Underlying	"ModelUndPrice"	The current model price of the option underlying.

Note: *If you do not have the corresponding* ***Market Data Subscription*** *(*p. **??***), '0' will be displayed if you request for live tick types above. Please refer to* ***Delayed Tick Types*** *if you are interested.*

API version 9.73.05 or higher is required to request option greeks data.

5.4.8.2.2 Generic Tick Types

A selection of **Generic Tick Types** are also supported in TWS RTD Server API. To request for any **Generic Tick Type**, you just need to specify the name of the generic tick type as the **Topic** string in the RTD formula.

For example, the below formula will request the 52-Week High price for IBM@SMART:

```
=RTD("Tws.TwsRtdServerCtrl"„"IBM@SMART", "Week52Hi")
```

See table below for a full list of currently supported **Generic Tick Types**:

Generic Tick Type Name	Topic String	Description	Generic Tick Required
Auction Volume	"AuctionVolume"	The number of shares that would trade if no new orders were received and the auction were held now.	225
Auction Imbalance	"AuctionImbalance"	The number of unmatched shares for the next auction; returns how many more shares are on one side of the auction than the other.	225
Auction Price	"AuctionPrice"	The price at which the auction would occur if no new orders were received and the auction were held now. The indicative price for the auction.	225
Regulatory Imbalance	"RegulatoryImbalance"	The imbalance that is used to determine which at-the-open or at-the-close orders can be entered following the publishing of the regulatory imbalance.	225
PL Price	"PlPrice"	The PL Price, also known as the Mark Price, is the current theoretical calculated value of an instrument. Since it is a calculated value, it will typically have many digits of precision.	232
Creditmanager Mark Price	"CreditmanMarkPrice"	Not currently available.	221
Creditmanager Slow Mark Price	"CreditmanSlowMark↩ Price"	Slow Mark Price update used in system calculations (same as Mark Price update in TWS Account Window -> Portfolio).	619
Call Option Volume	"CallOptionVolume"	Call option volume for the trading day.	100
Put Option Volume	"PutOptionVolume"	Put option volume for the trading day.	100
Call Option Open Interest	"CallOptionOpenInterest"	Call option open interest.	101
Put Option Open Interest	"PutOptionOpenInterest"	Put option open interest.	101
Option Historical Volatility	"OptionHistoricalVol"	The 30-day historical volatility (currently for stocks).	104
RT Historical Volatility	"RTHistoricalVol"	30-day real time historical volatility (Futures only).	411

Generic Tick Type Name	Topic String	Description	Generic Tick Required
Option Implied Volatility	"OptionImpliedVol"	A prediction of how volatile an underlying will be in the future. The IB 30-day volatility is the at-market volatility estimated for a maturity thirty calendar days forward of the current trading day, and is based on option prices from two consecutive expiration months.	106
Index Future Premium	"IndexFuturePremium"	The number of points that the index is over the cash index (Indeses only).	162
Shortable	"Shortable"	Describes the level of difficulty with which the contract can be sold short. See **Shortable** .	236
Fundamental Ratios	"Fundamentals"	Provides the available Reuter's Fundamental Ratios. See **Fundamental Ratios** .	258
Trade Count	"TradeCount"	Trade count for the day.	293
Trade Rate	"TradeRate"	Trade count per minute.	294
Volume Rate	"VolumeRate"	Volume per minute.	295
Last RTH Trade	"LastRthTrade"	Last Regular Trading Hours traded price.	318
IB Dividends	"IBDividends"	Contract's dividends. See **IB Dividends** .	456
Bond Factor Multipler	"BondMultiplier"	Not currenctly available.	460
Average Volume	"AvgVolume"	The average daily trading volume over 90 days (multiply this value times 100).	165
High 13 Weeks	"Week13Hi"	Highest price for the last 13 weeks.	165
Low 13 Weeks	"Week13Lo"	Lowest price for the last 13 weeks.	165
High 26 Weeks	"Week26Hi"	Highest price for the last 26 weeks.	165
Low 26 Weeks	"Week26Lo"	Lowest price for the last 26 weeks.	165
High 52 Weeks	"Week52Hi"	Highest price for the last 52 weeks.	165
Low 52 Weeks	"Week52Lo"	Lowest price for the last 52 weeks.	165
Short-Term Volume 3 Minutes	"ShortTermVolume3Min"	The past three minutes volume. Interpolation may be applied.	595

Generic Tick Type Name	Topic String	Description	Generic Tick Required
Short-Term Volume 5 Minutes	"ShortTermVolume5Min"	The past five minutes volume. Interpolation may be applied.	595
Short-Term Volume 10 Minutes	"ShortTermVolume10↩ Min"	The past ten minutes volume. Interpolation may be applied.	595
Futures Open Interest	"FuturesOpenInterest"	Total number of outstanding futures contracts (T↩ WS Build 965+ is required)	588
Average Option Volume	"AvgOptVolume"	Average volume of the corresponding option contracts (TWS Build 970+ is required)	105

By default, all **Generic Tick Types** are automatically requested. User just need to directly specify the **Topic** as the name of a generic tick type to populate the data to Excel.

In order to consume less data resource and make your market data request more efficient, you can directly specify the **Generic Tick Type** to be requested by defining string *"genticks=id1,id2,..."*.

For example, to request 52-Week High price, only **Generic Tick Type** = 165 is required. The below formula will only request **Generic Tick Type** = 165:

```
=RTD("Tws.TwsRtdServerCtrl"„"IBM@SMART", "Week52Hi", "genticks=165")
```

5.4.8.2.3 Delayed Tick Types

When live streaming market data is not availale because of missing **Market Data Subscription** (p. **??**), delayed data will be automatically relayed back. To request delayed data via RTD, you need to specify delayed tick types for the **Topic**. The table below shows a full list of available delayed tick types:

Tick Name	Topic String	Description
Delayed Bid Size	"DelayedBidSize"	Number of contracts (or lots) offered at the bid price.
Delayed Bid Price	"DelayedBid"	Highest bid price for the contract.
Delayed Ask Price	"DelayedAsk"	Lowest offer price for the contract.
Delayed Ask Size	"DelayedAskSize"	Number of contracts (or lots) offered at the ask price.
Delayed Last Price	"DelayedLast"	Last price at which the contract traded.
Delayed Last Size	"DelayedLastSize"	Number of contracts or lots traded at the last price.
Delayed High	"DelayedHigh"	High price for the day.
Delayed Low	"DelayedLow"	Low price for the day.
Delayed Volume	"DelayedVolume"	Trading volume for the day for the selected contract (Volume for US Stocks are quoted in lots. The actual number of shares in volume can be calculated by multiplying 100).
Delayed Close Price	"DelayedClose"	The last available closing price for the previous day. For US Equities, we use corporate action processing to get the closing price, so the close price is adjusted to reflect forward and reverse splits and cash and stock dividends.
Delayed Open Price	"DelayedOpen"	Today's opening price. The official opening price requires a market data subscription to the native exchange of a contract.
Delayed Last Timestamp	"DelayedLastTimestamp"	Delayed time of the last trade (in UNIX time) (TWS Build 970+ is required).

For example, the below formula will request the delayed bid price for IBM from Island exchange:

```
=RTD("Tws.TwsRtdServerCtrl",,"IBM@ISLAND", "DelayedBid")
```

Note: *Delayed tick types are 15-minute delayed. Requesting for live tick types without market data subscription will result in error message "Requested market data is not subscribed. Displaying delayed market data..."*

See more **Syntax examples** (p. **??**).

5.4.8.3 Connection Parameters

Since the TWS RTD Server API directly refers to the C# API Client, so it connects to TWS (or IB Gateway) the same as C# via the socket. The **Host IP Address**, **Socket Port** and **Client ID** are required parameters for initiating a socket connection.

- The **Host IP Address** is the IP address where your TWS is running on. For a local connection, local IP 127.0.0.1 can be used.
- The **Socket Port** is the port for socket connection. You can setup the host port in **TWS API Settings** (p. **??**), and you need to have your API connect to the same port as you setup in TWS.
- The **Client ID** is an identification for each API connection. TWS can maintain up to 32 API Clients connecting at the same time, and the Client ID is used to distinguish each connection. This was originally designed so that API users can have multiple API programs (i.e. clients) running at the same using different strategies to trade separately. Since the TWS RTD Server API is only provided for relaying real-time data, there is no need to use multiple client IDs.

The above three parameters are defaulted to the following values if not directly specified by the user:

- **Host** = "127.0.0.1" (i.e. the "localhost")
- **Port** = "7496"
- **ClientID** = Integer.MaxValue - 1

Simple Syntax supports several pre-defined **Connection Parameters** that can be specified as a separate string (i.e. String2, String3...) in the RTD formula:

- *"paper"*: use port=7497 for connection instead (7497 is the default port for paper TWS essions)
- *"gw"*: use port=4001 for connection instead (4001 is the default port for live IB Gateway sessions)
- *"gwpaper"*: use port=4002 for connection instead (4002 is the default port for paper IB Gateway sessions)

For example, to request High price for IBM@SMART while connecting to a TWS logged with a paper account via port 7497:

```
=RTD("Tws.TwsRtdServerCtrl",,"IBM@ISLAND", "High", "paper")
```

5.4.8.4 Additional Info

See also:

- **Complex Syntax**
- **Mixed Syntax**

5.4.9 Complex Syntax

Complex Syntax provides the most flexibility that it allows users to customize all formula strings individually, where each string only represent one single parameter. There is no rule for the sequence of the appearance of each parameter.

```
=RTD(ProgID, Server, String1, String2, String3...)
```

where

- **ProgID** = "Tws.TwsRtdServerCtrl"
- **Server** = "" (empty string)

For exmaple, the formula below will request the Ask Size for IBM@ISLAND:

```
=RTD("Tws.TwsRtdServerCtrl",,"sym=IBM", "sec=STK", "exch=ISLAND", "qt=Ask↩
Size")
```

The formula below will request the Bid price for IBM:

```
=RTD("Tws.TwsRtdServerCtrl",,"sym=ES", "sec=FUT", "exch=GLOBEX", "cur=USD",
"exp=201712", "qt=Bid")
```

5.4.9.1 Complex Syntax Strings

See table below for a full list available Complex Syntax strings:

Name	String Syntax	Description
Contract ID	"conid="	The unique contract ID generated by IB. Can be found at TWS **Contract Description** .
Symbol	"sym="	The contract symbol.
SecurityType	"sec="	The type of security, e.g. 'STK', 'FUT' and so on.
LastTradeDateOrContractMonth	"exp="	Format 'YYYYMMDD' is used for defining the Last Trade Date, while format 'YYYYMM' is used for defining the Contract Month.
Strike	"strike="	The strike price for an option contract.
Right	"right="	'C' or 'P' for an option contract.
Multiplier	"mult="	The contract multiplier.

Name	String Syntax	Description
Exchange	"exch="	The exchange where to get market data from. For equities, 'SMART' means top data from all possible exchanges.
PrimaryExchange	"prim="	The primary exchange of the contract. It is mostly specified when an contract ambiguity occurs for equity symbols that are listed on multiple exchanges.
Currency	"cur="	The currency the contract is traded in.
LocalSymbol	"loc="	The local symbol of the contract. Note the Local Symbol is mostly used for futures and options, and is different from the Symbol.
TradingClass	"tc="	The trading class of the contract.
Combo	"cmb="	Combo contract has to be defined using Complex Syntax or Mixed Syntax. The syntax for defining the combo is: "cmb=<conid1>#<ratio1>#<action1>#<exchange1>;<conid2>#<ratio2> ,where combo legs are separated by ';' and individual leg parameters are separated by '#'. See more **Spread Samples** .
DeltaNeutralContract	"und="	Delta-Neutral Contract. The syntax for defining the delta-neutral contract is: "und=<<conid>#<delta>#<price>" , where delta-neutral contract parameters are separated by '#'.
MktDataOptions	"opt="	Currently not supported.
GenericTickList	"genticks="	A comma separated Ids of available Generic Tick Types.
Topic	"qt="	Topic of market data request.
Host	"host="	Host IP address.
Port	"port="	Socket port.
ClientId	"clientid="	The client ID for socket connection. Note that the client ID is used for identify multiple simultaneous API connections to the same TWS. It was originally designed for API users who would like to manage their strategies separately from different API programs. Since the TWS RTD Server API is currently only supported for real-time market data, there is no need to use multiple client IDs.

5.4.9.2 Additional Info

See also:

- **Simple Syntax**
- **Mixed Syntax**

5.4.10 Mixed Syntax

The Complex Syntax can be mixed with **Simple Syntax** with only one restriction that **Ticker** string must be the first string (i.e. String1) that appears in the RTD formula.

```
=RTD(ProgID, Server, Ticker, String2, String3...)
```

where

- **ProgID** = "Tws.TwsRtdServerCtrl"
- **Server** = "" (empty string)

For example, the formula below will request the Bid Price for IBM while connecting through a customized Host, Port and Client ID:

```
=RTD("Tws.TwsRtdServerCtrl",,"IBM@SMART", "host=1.2.3.4", "port=1234", "client↩
Id=1", "Bid")
```

See more **Syntax Samples** (p. **??**).

5.4.10.1 Additional Info

See also:

- **Simple Syntax**
- **Complex Syntax**

5.4.11 TWS RTD Server Samples

This page is provided as a demonstration of RTD formulas categorized by security type as well as syntax type. Make sure to get yourself familiar with the various available Syntaxs for RTD formula before looking into the samples.

- **Simple Syntax**
- **Complex Syntax**
- **Mixed Syntax**

5.4.11.1 Forex Pairs

5.4.11.1.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"EUR.USD/CASH", "Bid")
```

Comment: Forex Ticker is defined in format "CURRENCY1.CURRENCY2/CASH".

5.4.11.1.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"sym=EUR","cur=USD", "exch=IDEALPRO", "sec=CAS↩
H", "qt=Bid")
```

*Comment: For Complex Syntax, Forex **Symbol** is defined as the foreign currency, and the **Currency** is defined as the base currency.*

5.4.11.1.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"EUR.USD/CASH", "Bid", "port=1234", "client↩
Id=1")
```

5.4.11.2 Stocks

5.4.11.2.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"IBM")
```

*Comment: Default values are used: **Exchange** = "SMART", **Currency** = "USD", **Security Type** = "STK", **Topic** = "Last".*

```
=RTD("Tws.TwsRtdServerCtrl",,"IBM@ISLAND", "Bid")
```

*Comment: Specifying the **Exchange** directly means requesting data from that exchange specifically.*

```
=RTD("Tws.TwsRtdServerCtrl",,"BMO@SMART/////CAD", "Bid")
```

Comment: Currency = "CAD" is needed for BMO listed on TSE, but the rest of the fields can be left out as blank.

```
=RTD("Tws.TwsRtdServerCtrl",,"ABG.P@SMART/////EUR", "Close")
```

Comment: Stock symbols that contain a '.' are supported in Simple Syntax as well.

```
=RTD("Tws.TwsRtdServerCtrl",,"MSFT@SMART/ISLAND", "Ask")
```

*Comment: For certain smart-routed stock contracts that have the same **Symbol**, **Currency** and **Exchange**, you would also need to specify the **PrimaryExchange** attribute to uniquely define the contract. This should be defined as the native exchange of a contract, and is good practice for all stocks.*

5.4.11.2.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"sym=IBM", "sec=STK", "exch=SMART", "cur=USD",
"qt=Volume")
```

*Comment: Generally speaking, using the **Symbol**, **SecurityType**="STK", **Currency** and **Exchange** is sufficient to define a stock.*

```
=RTD("Tws.TwsRtdServerCtrl",,"sym=MSFT", "sec=STK", "exch=SMART", "cur=USD",
"prim=ISLAND", "qt=Open")
```

*Comment: Specifying the **PrimaryExchange** as a seprate string to resolve contract ambiguity.*

5.4.11.2.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"MSFT@SMART", "prim=ISLAND", "qt=High", "paper")
```

Comment: Use the pre-defined string "paper" to connect to the default port 7497 for paper TWS sessions.

5.4.11.3 Indexes

5.4.11.3.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"SPX@CBOE//IND", "Last")
```

*Comment: Default **Currency** = "USD" is used.*

```
=RTD("Tws.TwsRtdServerCtrl",„"DAX@DTB//IND////EUR", "Last")
```

5.4.11.3.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"sym=INDU","cur=USD", "exch=NYSE", "sec=IND",
"qt=Close")
```

5.4.11.3.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"DAX@DTB//IND", "cur=EUR", "qt=Last", "host=1.↩
2.3.4")
```

5.4.11.4 CFDs

5.4.11.4.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"IBDE30@SMART//CFD////EUR", "Bid")
```

5.4.11.4.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"sym=IBDE30","cur=EUR", "exch=SMART", "sec=CF↩
D", "qt=ASK")
```

5.4.11.4.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"IBDE30@SMART//CFD", "cur=EUR", "Bid", "gw")
```

Comment: Use the pre-defined string "gw" to connect to the default port 4001 for live IB Gateway sessions.

***Note:** Only Index CFD data can be directly queried via the API, but not equity CFD. Please directly request data for the underlying equity if you need data for an equity CFD contract.*

5.4.11.5 Futures

5.4.11.5.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"ES@GLOBEX//FUT/201712///USD", "Bid")
```

*Comment: Use underlying **Symbol** and **LastTradeDateOrContractMonth** to define futures contract.*

5.4.11.5.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"loc=ESZ7","cur=USD", "exch=GLOBEX", "sec=FUT",
"qt=Ask")
```

*Comment: The **LastTradeDateOrContractMonth** and underlying **Symbol** can be replaced with the contract's own symbol, also known as **LocalSymbol** (named as Symbol within the TWS' Contract Description dialog). Local Symbol is not available in Simple Syntax.*

5.4.11.5.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"DAX@DTB//FUT/201706///EUR", "mult=5", "Low")
```

*Comment: For futures that have multipler **Multipliers** (e.g. DAX has 5 and 25), Simple Syntax is not adequate to define the contract uniquely. Mixed Syntax can help to add addition specification for the **Multiplier**.*

5.4.11.6 Options

5.4.11.6.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"GOOG@SMART//OPT/20170421/C/835/USD", "Bid")
```

*Comment: Use **Symbol**, **LastTradeDateOrContractMonth**, **Right** and **Strike** to define options contract.*

5.4.11.6.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"loc=C DBK DEC 20 1600", "cur=EUR", "exch=DTB",
"sec=OPT", "qt=Close")
```

*Comment: Use **LocalSymbol** to define options contract.*

5.4.11.6.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"SANT@MEFFRV//OPT/20190621/C/7.5/EUR", "mult=100",
"tc=SANEU", "Close")
```

*Comment: For options that have multipler **Multipliers** or **TradingClasses**, Simple Syntax is not adequate to define the contract uniquely. Mixed Syntax can help to add addition specifications for **Multipler** and **TradingClass** properly.*

5.4.11.7 Futures Options

5.4.11.7.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"ES@GLOBEX//FOP/20180316/C/1000/USD", "Close")
```

Comment: Futures Options follow the same rule as conventional option contracts.

5.4.11.7.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"loc=ESH8 C1000", "cur=USD", "exch=GLOBEX",
"sec=FOP", "qt=Close")
```

5.4.11.7.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"ES@GLOBEX//FOP/20180316/C/1000/USD", "mult=50",
"tc=ES", "Close")
```

5.4.11.8 Bonds

5.4.11.8.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"912828C57@SMART//BOND", "Bid")
```

*Comment: Bonds can be specified by defining the **Symbol** as the CUSIP. **Currency** = "USD" is used as default here.*

5.4.11.8.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"sym=912828C57","cur=USD", "exch=SMART", "sec=↩
BOND", "qt=Bid")

=RTD("Tws.TwsRtdServerCtrl",,"conid=147554578", "exch=SMART", "qt=Ask")
```

*Comment: Bonds can also be defined with the **ConId** and **Exchange** as with any security type.*

5.4.11.8.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",,"912828C57@SMART, "sec=BOND", "Bid", "gwpaper")
```

Comment: Use the pre-defined string "gwpaper" to connect to the default port 4002 for paper IB Gateway sessions.

5.4.11.9 Mutual Funds

5.4.11.9.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"VINIX@FUNDSERV//FUND", "Close")
```

5.4.11.9.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"sym=VINIX","cur=USD", "exch=FUNDSERV", "sec=F↩
UND", "qt=Close")
```

5.4.11.9.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"VINIX@FUNDSERV//FUND", "Bid", "host=1.2.3.4",
"port=1234", "clientId=1")
```

5.4.11.10 Commodities

5.4.11.10.1 - Simple Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"XAUUSD@SMART//CMDTY", "Bid")
```

5.4.11.10.2 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"sym=XAUUSD","cur=USD", "exch=SMART", "sec=CMD↩
TY", "qt=Ask")
```

5.4.11.10.3 - Mixed Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"XAUUSD@SMART//CMDTY", "Last", "port=1234",
"clientId=1")
```

5.4.11.11 Spreads

Spread contracts, also known as combos or combinations, combine two or more instruments. To define a combination contract it is required to know the Contract ID of the combo legs. The **ConId** can be easily found in the **Contract Description** page in TWS.
The spread contract's symbol can be either the symbol of one of the contract legs or, for two-legged combinations the symbols of both legs separated by a comma as shown in the examples below.

Simple Syntax is not sufficient to define spread contracts. You need to use either **Complex Syntax** or **Mixed Syntax** . As a reminder, here is the string formula for defining the Combo Legs:

"cmb=<conid1>#<ratio1>#<action1>#<exchange1>;<conid2>#<ratio2>#<action2>#<exchange2>;"

5.4.11.11.1 Stock Spread

Buy 1 IBKR@SMART + Sell 1 MCD@SMART:

5.4.11.11.1.1 - Complex Syntax

```
=RTD("Tws.TwsRtdServerCtrl",„"sym=IBKR,MCD", "exch=SMART", "cur=USD", "sec=↩
BAG", "cmb=43645865#1#BUY#SMART;9408#1#SELL#SMART;", "Bid")
```

5.4.11.11.1.2 - Mixed Syntax

```
=RTD("tws.twsrtdserverctrl",„"IBKR,MCD@SMART//BAG////USD", "cmb=43645865#1#↩
BUY#SMART;9408#1#SELL#SMART;", "Bid")
```

Note: *EFPs are simply defined as a bag contract of stock and corresponding SSF with a ratio of 100:1.*

5.4.11.11.2 Futures Spread

Buy 1 VXJ7@CFE + Sell 1 VXK7@CFE:

5.4.11.11.2.1 - Complex Syntax

```
=RTD("tws.twsrtdserverctrl",„"sym=VIX", "exch=CFE", "cur=USD", "sec=BAG",
"cmb=249139906#1#BUY#CFE;252623425#1#SELL#CFE;", "Bid")
```

5.4.11.11.2.2 - Mixed Syntax

```
=RTD("tws.twsrtdserverctrl",„"VIX@CFE//BAG////USD", "cmb=249139906#1#BUY#CF↩
E;252623425#1#SELL#CFE;", "Bid")
```

5.4.11.11.3 Options Spread

Buy 1 DBK May19'17 15 CALL @DTB + Sell 1 DBK May19'17 16 CALL @DTB:

5.4.11.11.3.1 - Complex Syntax

```
=RTD("tws.twsrtdserverctrl",„"sym=DBK", "exch=DTB", "cur=EUR", "sec=BAG",
"cmb=270579950#1#BUY#DTB;270579957#1#SELL#DTB;", "Bid")
```

5.4.11.11.3.2 - Mixed Syntax

```
=RTD("tws.twsrtdserverctrl",„"DBK@DTB//BAG////EUR", "cmb=270579950#1#BUY#DT↩
B;270579957#1#SELL#DTB;", "Bid")
```

5.4.11.11.4 Inter-Commodity Futures Spread

For Inter-Commodity futures, the 'Local Symbol' field in TWS is used for the 'Symbol' field in the TWS **Contract Description** (p. **??**).

Buy 1 CL May'17 @NYMEX + Sell 1 BZ Jun'17 @NYMEX:

5.4.11.11.4.1 - Complex Syntax

```
=RTD("tws.twsrtdserverctrl",,"sym=CL.BZ", "exch=NYMEX", "cur=USD", "sec=BA↩
G", "cmb=55977404#1#BUY#NYMEX;55807026#1#SELL#NYMEX;", "Bid")
```

5.4.11.11.4.2 - Mixed Syntax

```
=RTD("tws.twsrtdserverctrl",,"CL.BZ@NYMEX//BAG///USD", "cmb=55977404#1#BU↩
Y#NYMEX;55807026#1#SELL#NYMEX;", "Bid")
```

Note: *Please be mindful of the fact that Inter-commodity Futures contracts are spread contracts offered by the exchange directly, and the contract definition is different from regular combo contracts. Please make sure all the contract attributes are specified in accordance with TWS* ***Contract Description*** *page.*

5.4.11.12 Sample Spreadsheet

A sample RTD spreadsheet is provided within the API installation directory. By default, you will be able to find it under *C:\TWS API\samples\ExcelTwsRtdServer.xls*.

5.4.12 Change Data Refresh Rate

It is important to mention that our real time market data is not tick-by-tick, meaning you will not obtain every single price movement happening in the market. Instead, real time data is given as snapshots generated at a fixed given pace:

Product	Frequency
Stocks, Futures and others	250 ms
US Options	100 ms
FX pairs	5 ms

Microsoft RTD interface has a `ThrottleInterval` property that determines the interval between data refreshes. By default, the value is set to 2000 milliseconds, which means Excel waits at least 2000 milliseconds between checks for updates. You are able to manually change the Throttle Interval to a smaller value* so as to increase the refresh rate of real time data.

The easiest way to change the ThrottleInterval property is through VBA:

1. In Excel, go to the Visual Basic Editor window by pressing **Alt_F11**.
2. On the Visual Basic Editor window, click on **View ->** **Immediate Window** or hold **Ctrl_G** to open the **Immediate Window**.
3. On the **Immediate Window**, type in the following code and then click **Enter**:

   ```
   Application.RTD.ThrottleInterval=250
   ```

4. To verify that it is set correctly, type this line of code on the **Immediate Window** and click **Enter**:

   ```
   ?  Application.RTD.ThrottleInterval
   ```

5. Verify the next line should display 250. If this value is changed, the new value will persist when Microsoft Excel is restarted.

∗***Warning:*** *As the* `ThrottleInterval` *is lowered, updates can come in so frequently that Excel is continuously updating values and doing calculations, Excel might end up in a state where it never gives the user a chance to do anything, effectively getting in a hung state. If this happens, set the Excel throttle interval higher.*

Source: Microsoft Real-Time Data: Frequently Asked Questions `How Do I Configure the RTD Throttle Interval in Excel?`.

5.4.13 Troubleshooting Common Errors for RTD

5.4.13.1 Troubleshooting Common Errors

- **TwsRtdServer error: Cannot connect to TWS.**
- **TwsRtdServer error: No security definition has been found for the request.**
- **TwsRtdServer error: The contract description specified for <SYMBOL> is ambiguous.**
- **TwsRtdServer error: Requested market data is not subscribed. Displaying delayed market data...**(p. **??**)
- **TwsRtdServer error: Error validating request:-'zd':cause - Incorrect generic tick list of**
- **Some data show '0' when requesting data for many securities**

5.4.13.1.1 TwsRtdServer error: Cannot connect to TWS.

This error message is most likely triggered because your TWS has not been configured properly for API socket connection. Please make sure to **Enable ActiveX and Socket Client** settings in your TWS. Also bear in mind that TWS Rtd Server connects to socket port 7496 by default. You will see the above error message if the socket port configured in your TWS API settings does not match what RTD is trying to connect to. See more details in **What You Will Need** .

5.4.13.1.2 TwsRtdServer error: RTD Server disconnects from TWS such that cells stop updating

This can occur if the API message rate of 50 messages/second is exceeded. No more than 50 messages can be sent from an API application such as RTD Server to TWS per second (this does not include messages in the opposite direction). Each ticker subscription request and subscription cancellation request corresponds to 1 message. If the 50 messages/second rate is exceeded, TWS will eventually close the connection. So for constructing an RTD spreadsheet with more than 50 tickers, it must be built to only make at most 50 new subscriptions or cancellations per second.

5.4.13.1.3 TwsRtdServer error: No security definition has been found for the request.

This error message is triggered to indicate the contract definition provided in your RTD formula cannot be found by TWS. Usually it is caused by incorrect contract attribute definitions or typo. You are suggested to refer to some **Syntax Samples** and find out the issue in your RTD formula.

5.4.13.1.4 TwsRtdServer error: The contract description specified for <SYMBOL> is ambiguous.

This error message indicates the contract definition provided by you does not uniquely define one single contract. It is mostly triggered for stock symbols, such as MSFT and CSCO, that are listed on multiple primary exchanges. Specifying the **PrimaryExchange** will resolve the issue. Please refer to **Syntax Sample for MSFT** .

5.4.13.1.5 TwsRtdServer error: Requested market data is not subscribed. Displaying delayed market data...

The error message is displayed when you are trying to request for a live tick type, while only delayed data is available due to missing **Market Data Subscription** (p. **??**). You need to either subscribe to market data pacakegs via Account Management, or request for **Delayed Tick Types** instead.

5.4.13.1.6 TwsRtdServer error: Error validating request:-'zd':cause - Incorrect generic tick list of <list>

As explained in **What You Will Need** , TWS Build 963+ is required because there are some generic tick types requested by RTD by default that are not supported in older TWS builds. `Upgrade TWS` to any build above 963 will resolve the issue.

5.4.13.1.7 Some data show '0' when requesting data for many securities

This is most likely because you have exceeded the limit of **Market Data Lines** . You can verify if this is reason by going to TWS and hold **Ctrl_Alt_=** at the same time. This should show a small pop-up window and indicate the maximum allowed market data lines as well as the currently subscribed top market data count. If you are subscribed to more than the maximum allowed, some of the data point will show '0'.

5.5 Dynamic Data Exchange

The Dynamic Data Exchange protocol is a method of inter-process communication developed by Microsoft to establish communication between Windows applications running on the same computer. The DDE API is available for Windows computers to create a means of communication between Microsoft Excel and TWS or IB Gateway.

Requirements for the DDE API

- Windows OS
- **32 bit version** of TWS or IB Gateway
- Microsoft Excel
- TWS API installed to the C: drive
- Excel must be set to the US settings for commas and periods. That is, commas denote thousands and periods denote decimals.

Interactive Brokers does not offer any programming assistance and therefore it is strongly advised to anyone willing to use any of the TWS DDE API to become familiar with the technologies involved such as the DDE protocol and VBA.

You can get started by following the **Tutorial** and learn how to use DDE to communicate with TWS.

Limitations specific to DDE

- DDE does not have the full range of tick types available to the socket-based APIs
- DDE **requires live market data** in TWS to receive either streaming or historical market data. It is not compatible with delayed data.
- Other applications running DDE in the background can interfere with Excel DDE and cannot be used simultaneously.
- A DDE command string can be at most **255** characters. For that reason very lengthy commands such as those involving four-legged spreads are not available.

5.5.1 Troubleshooting Excel DDE

- **Error: "Remote Data Not Accessible..." causes**

When this error is encountered in using a DDE formula in an Excel spreadsheet it indicates a general problem in the DDE connection between Excel and TWS. This can have any of the following causes:

- **1)** TWS is not 32 bit. By default 64 bit TWS is installed on modern computers. It is necessary to select the 32 bit version when downloading to use the DDE API.
- **2)** The API is not installed to the C: drive. The API can be downloaded from `http://interactivebrokers.github.io/` . After installation, there should be a folder "TWS API" on the C: drive. If the API installs to any other drive this can cause issues.
- **3)** "Enable DDE" is not checked in the TWS (or IB Gateway Global Configuration). In TWS this setting is at Global Configuration -> API -> Settings -> "Enable DDE".

- **4)** The incorrect username is input into a formula in Excel. The username in the spreadsheet must match the username used to login to TWS. For paper accounts, it must be the paper account username and not the live username (every paper account has its own username distinct from the live username).

(5) In rare cases it is necessary to run Excel as Administrator to use DDE.

Note: Other programs running on the same computer which use DDE can interfere with the communication between Excel and TWS and cause Excel to 'hang' or 'freeze' after making the initial request. The only solution to this problem is to close other programs using DDE one-by-one to find the culprit. Programs which are known to cause this issue include Google Chrome, Microsoft OneNote, Skype and Adobe Creative Cloud.

- The DDE API is now considered completed and will not have additional functionality that is added to socket-based APIs.

5.5.2 Limitations of Microsoft Excel APIs

By design, Microsoft Excel gives precedence to the user interface over the data connection to other applications. For that reason, Excel only receives updates when it is in a 'ready' state, and may ignore data sent for instance when a modal dialogue box is displayed to the user, a cell is being edited, or Excel is busy doing other things. A new Excel Real Time Data server (RTD) API is being introduced to help address some of these deficiencies, but unfortunately these limitations are inherent to Excel as a trading application and not specific to an API technology.

5.5.3 Tutorial

5.5.3.1 What You Will Need

This tutorial has been developed using Excel 2010 around the 9.72 version of the TWS API components. This version needs to be downloaded and installed before proceeding. You can download this version from `http://interactivebrokers.github.io/`.
The TWS must be up and running while using the DDE connection to TWS. DDE client connectivity must also be enabled via Edit -> Global Configuration -> API -> Settings: Enable DDE clients as indicated in the picture below:

5.5.3.2 Ddedll.dll Error

TWS users running add-on applications in Excel using DDE on a Windows 64-bit operating systems may receive the following error message upon log in or when enabling the DDE client:

Please follow the instruction `How to fix the error: "Ddedll.dll file missing or out of date"` to resolve the error before going forward to Tutorials.

5.5.3.3 Tutorial Topics

- **Real Time Data with Excel**
- **Historical Data with Excel**
- **DDE Formula Reference**

5.5.3.4 Real Time Data with Excel

One of the most common inquiries we receive at Interactive Brokers is about export data from the TWS into Excel. Since the TWS does not have functionality to export intra-day, customers are often directed at the TWS API and its Excel sheets in particular. Customers, however, need to be aware of the fact that the distributed sheets are not tools to be used on a daily basis. All our sample applications are merely a demonstration of the API capabilities aimed at experienced programmers who will in turn use them as a reference to develop more complex and robust systems.

It is possible to view live quotes for multiple products updating real time within Excel. Requests via the TWS DDE API are nothing but Excel formulas (DDE data links) each of them serving a very specific purpose. Market Data retrieval requires at least two different DDE links: one to start the market data subscription and another one which will be receiving the specific tick type.

This document is a brief tutorial explaining the process behind market data retrieval through MS Excel via the TWS DDE API. All the VBA code is kept to a minimum and its purpose is merely illustrative.

5.5.3.4.1 Requisites

Please make sure you have already acquired **What You Will Need** before going forward in this tutorial.

5.5.3.4.2 Preparing the request

The formula to start the request will need to provide the TWS with enough information so that the TWS can unambiguously identify which instrument we are interested in. As a first example, we will request FX market data (EUR.USD). The resulting formula will therefore be:

=Ssample123|tik!'id1?req?EUR_CASH_IDEALPRO_USD_∼/'

Note that sample123 will have to be replaced with the exact same user name entered when logging into the TWS. This applies for all subsequent DDE links shown in this guide.

For more contract definition samples via DDE, please refer to **How to Find the Definition of a Contract** . If the

above formula is copied into any cell of a spreadsheet, the cell should automatically display 0 on it.

After we have done this, the TWS will be aware that a DDE link is requesting EUR.USD data.

5.5.3.4.3 Receiving the data

Once the TWS recognizes our DDE link trying to pull EUR.USD data, we can proceed to read it. We are currently interested in knowing the bid price of the EUR.USD FX pair, therefore we need to make use of:

=Ssample123|tik!id1?bid

Pasting the above formula into the same sheet will have a more exciting result:

The value displayed on cell D2 is the exact same value the TWS displays for the EUR.USD bid price (at the moment of writing this document) and will keep updating as long as the request is active.

5.5.3.4.4 Understanding the Formulas

As it was already mentioned, the first formula will ask the TWS to open a DDE channel through which we can obtain EUR.USD data while the second one is merely pulling the bid price for it. To understand the formulas' syntax refer to the **Real Time Data** section from the **DDE Formula Reference** page.

5.5.3.4.5 Obtain the Last Available Error

Unfortunately things do not always work as expected. The slightest error in the DDE link or the contract description that you provide will prevent you from receiving the market data from TWS. The first and most obvious step in solving this problem is to make sure that your DDE links are correct and contain no spelling errors or typographical errors such as unwanted spaces or characters.

If the formula is correct but you are still not able to see any data, you can ask the TWS about any errors generated in response to your request. In most cases, TWS will be able to point us in the right direction.

TWS can only remember the most recent error. This is very important to remember because your Excel spreadsheet will often have many active requests with multiple possibilities for errors. Be sure that all previous requests are working as expected before creating a new one. This will help identify any problem.

There are three formulas that you need to use. Enter each formula into its own cell in your Excel worksheet:

Formula	Description
=S[twsuser]|err!id	Obtains the failed request's unique ID.
=S[twsuser]|err!errorCode	The error code.
=S[twsuser]|err!errorMsg	The description of the error.

Let's look at an example. In the following figure, the real time data request formula's symbol has been intentionally modified to EUE instead of EUR. We've entered the three error formulas into three separate cells. We will receive an error for that request (the request ID is 1), along with the error code and description (No security definition has been found for the request). This error means that the contract for which we are requesting data cannot be found. In other words, the description of the contract in the DDE link is wrong.

Once you know what caused the error, you should clear the error formulas first, and then correct the original DDE link. When you do this, you will notice that the error formulas will return a 0 value:

5.5.3.4.5.1 Why is it important to first clear the error formula before correcting our request?

We mentioned earlier on that TWS will hold the last available error message of the last failed request. This implies that TWS will remember that, as in our example, the request identified with id X, has an error associated to it. It is very tempting to simply correct the typo in our request. However, this will create an "orphan" error in TWS and this "Can't find EId" error will also be sent to your Excel worksheet, as shown in the following image. This orphan error is basically TWS saying "I cannot find an error for this request."

You might think that you can easily ignore this error. Imagine, however, that you have many DDE links in your Excel worksheet and one of them resulted in a "no security definition has been found" error. Later, another link in your sheet causes the "Can't find EId" error to appear. You will only be able to see the last error, which is not really telling you much about why your Excel worksheet is not working as you expect. While this logic applies to all errors, this last error can be particularly misleading.

REMEMBER: Be sure that all of your previous requests are working as expected before moving to the next one.

5.5.3.5 Historical Data with Excel

In the previous tutorial, we showed you how to request real time quotes from TWS using the DDE TWS API. In this tutorial, we will show you how to request historical data from TWS, although the process for doing so is slightly more complicated. You will need to add some simple Visual Basic (VBA) code to your Excel worksheet to obtain the data.

5.5.3.5.1 Requisites

Please make sure you have already acquired **What You Will Need** before going forward in this tutorial.

5.5.3.5.2 Preparing the request

Just as with real time data, historical data requests need first to ask the TWS to "prepare" the data we are interested in. The TWS needs to know not only the specific instrument but also:

- The ending date and time from which we want to collect the data, formatted as: **yyyymmdd hh:mm:ss**.
- The time duration comprising the data from the ending date going back in time.
- The bar size (IB provides historical data in open, high, low and close bar data format).
- The type of data (i.e. MIDPOINT, TRADES, etc.).
- Whether we want data generated during regular trading session or not.
- The date format in which each bar's time and date will be presented.

The formula to be used for historical data requests is:

=[twsuser]|hist!'id[requestId]?req?[symbol]_[type]_[exchange]_[currency]_~/[yyyymmdd]singleSpace[HH]singleColon[mm]singleColon[ss]_[duration amount]singleSpace[duration unit]_[bar size]_[rth only?]_[what to show]_[date format]'

Attribute	Description
twsuser	The username with which you logged into TWS.
requestId	The request's unique identifier (any positive integer).
symbol	The instrument's symbol.
type	The type of instrument.
exchange	The instrument's exchange.
currency	The instrument's currency (USD).
yyyymmdd hh:mm:ss	End date for the historical data query.
duration amount	The number of time units for the duration time.
duration unit	The duration's time unit.
bar size	The bar size.
rth only	Set to 1 to obtain only data generated during regular trading hours (RTH), or set to 0 to get all data generated during and outside of of RTH.
what to show	The type of data: MIDPOINT, TRADES, BID, ASK, etc.
data format	Set to 1 to format the resulting bars' date as yyyymmss hh:mm:ss. Set to 2 to express the resulting bars' time as the number of seconds since 1970.

5.5.3.5.2.1 How to Handle Spaces and Colons in the Formula

Our DDE links cannot contain certain special characters such as spaces or colons, but you will need to use these characters in your DDE formula. To overcome this limitation, we have provided keywords that you can use in place of the actual special character: singleSpace and singleColon. For example, if you want to specify an end date and time such as March 2, 2015 at 23:59:59 in the format specified above, you would then enter:

20150302 23:59:59

This translates into:

20150302**singleSpace**23**singleColon**59**singleColon**59

This applies to all cases in which you need spaces or colons in the DDE formula. This is particularly important when describing futures or options contracts because you can then use their local symbols, which often include spaces. For example, the DBK futures contract expiring on May 2015 has a local symbol **DBKG MAY 15** which you would provide as:

DBKG**singleSpace**MAY**singleSpace**15

5.5.3.5.2.2 Enter the Historical Data Request

Let's continue with our historical data request. As an example, try to pull MIDPOINT historical data for the EUR.USD currency pair prior to February 27th 2015 at 23:59:59 in thirty minutes bars (9), for a duration of one day (1 D). The correct formula for this request is:

=S**sample123**|hist!'id**4**?req?**EUR_CASH_IDEALPRO_USD_**~/**20150227**singleSpace**23**singleColon**59**single↩
Colon**59_1**singleSpace**D_9_MIDPOINT_1_1**'

Copy the above formula into an empty cell in your Excel worksheet. Notice that the cell displays PROCESSING, which, if everything proceeds without error, will change into RECEIVED":

At this point, you have just told TWS that you want our EUR.USD historical data and TWS replied that the data has been received from the server and is ready to be viewed.

This is where the process becomes slightly complicated because, unlike real time market data, where each incoming price is obtained using a very specific formula, you will not fetch each bar one by one with a formula (this is quite fortunate since we could be expecting hundreds of bars!). Instead, you will read all the bars together using a single DDE request and then display them in your worksheet with the help of some VBA code. For purposes of simplicity, we will keep the coding to minimum.

In the next steps, we will briefly describe how to add a button to a spreadsheet for the sake of completeness but remember that it is out of the scope of IB's support to provide any assistance on using Excel.

5.5.3.5.3 Receiving the data

- **Receiving the Data - Add a Button**
- **Receiving the Data - Add the Code**

5.5.3.5.4 Understanding the Formula

To understand the formulas' syntax, please refer to the **Historical Data** section from the **DDE Formula Reference** page.

For more contract definition samples via DDE, please refer to **How to Find the Definition of a Contract** .

5.5.3.5.5 Receiving the Data - Add a Button

In this step, you will add a button to your blank worksheet which, once the TWS has replied to your historical data request with the “RECEIVED” status, will help you manually invoke the VBA routines which pull the historical data from TWS.

First, open the Developer tab in Excel and click on the Button form control:

Next, click anywhere in your spreadsheet to place the button. The Assign Macro dialog opens; this is where you associate a VBA macro with your button. Name your function **fetchHistoricalData** and then click the New button in the dialog.

Excel automatically opens the VBA editor, which displays the skeleton of the newly-created macro.

In the next step, you will add the code to the macro you just created.

- **Receiving the Data - Add the Code**

5.5.3.5.6 Receiving the Data - Add the Code

Here are the routines which will finally obtain the data from the TWS:

```
Sub fetchHistoricalData()

    'This variable will store the incoming data

    Dim TheArray() As Variant

    'Fetch the data from the TWS...

    '(Replace sample123 with your own TWS username!)

    TheArray = getData("Ssample123", "hist", "id4?result")

    '... and pass the result into another function which will populate the sheet

    Call populate(TheArray)

End Sub


'This function triggers a DDE request and returns its response

Function getData(serverName, topic, request)

    Dim chan As Integer

    'Initiate the DDE channel

    chan = Application.DDEInitiate(serverName, topic)

    'Perform the request

    getData = Application.DDERequest(chan, request)

    'Terminate the channel

    Application.DDETerminate chan

End Function
```

```
'Populate our blank sheet with the incoming data

Sub populate(ByRef TheArray() As Variant)

'Watch out for empty possible errors and handle properly.

On Error GoTo ErrHandler

    For i = 1 To UBound(TheArray)

        Range("F" & i + 1).Value = TheArray(i, 1)

        Range("G" & i + 1).Value = TheArray(i, 2)

        Range("H" & i + 1).Value = TheArray(i, 3)

        Range("I" & i + 1).Value = TheArray(i, 4)

        Range("J" & i + 1).Value = TheArray(i, 5)

        Range("K" & i + 1).Value = TheArray(i, 6)

        Range("L" & i + 1).Value = TheArray(i, 7)

        Range("M" & i + 1).Value = TheArray(i, 8)

        Range("N" & i + 1).Value = TheArray(i, 9)

    Next

ErrHandler:

    Exit Sub

End Sub
```

The fetchHistoricalData method invokes the getData function passing in:

- The DDE server name, which is your TWS username prefixed with a capital S
- The DDE "topic" for historical data: "hist"
- A third parameter which is just the remaining fragment of the DDE link: id[requestId]?result

The third parameter contains the request ID you used in the requesting formula (4). Remember this same procedure from the previous tutorial when you requested real time data. Your request/retrieve formulas both need to include the exact same ID.

If you correctly entered the code into your macro in the VBA editor as shown above, your Excel worksheet should look very similar to the image below. (Note that we have changed the button label to **Historical** from its default value).

Just after the data is retrieved from TWS, the requesting formula will change its output to FINISHED.

It is very important for to wait until the request formula's output changes from PROCESSING to RECEIVED before you try to pull the actual data from TWS. If the cell displays PROCESSING for too long, then it is very likely there was an error in your request. If this happens, make use of the error retrieval formulas explained in **Obtain the Last Available Error** (p. **??**).

The next step is to **Understand the Formula** (p. **??**).

5.5.3.6 DDE Formula Reference

5.5.3.6.1 Real Time Data

Attribute	Description
twsuser	The user name with which you logged into the TWS
requestId	The request unique identifier (any positive integer)
symbol	The contract symbol
sectype	The kind of contract
exchange	The exchange from which we want to pull the data from
currency	The contract currency

5.5.3.6.1.1 Initial Request

=S[twsuser]|tik!'id[reqestId]?req?[symbol]_[sectype]_[exchange]_[currency]_~/'

5.5.3.6.1.2 Data Reception

=S[twsuser]|tik!id[requestId]?bidSize

=S[twsuser]|tik!id[requestId]?bid

=S[twsuser]|tik!id[requestId]?ask

=S[twsuser]|tik!id[requestId]?askSize

=S[twsuser]|tik!id[requestId]?last

=S[twsuser]|tik!id[requestId]?lastSize

=S[twsuser]|tik!id[requestId]?high

=S[twsuser]|tik!id[requestId]?low

=S[twsuser]|tik!id[requestId]?volume

=S[twsuser]|tik!id[requestId]?close

=S[twsuser]|tik!id[requestId]?bidImpliedVol

5.5.3.6.1.3 Option-specific Ticks

=S[twsuser]|tik!id[requestId]?bidDelta

=S[twsuser]|tik!id[requestId]?askImpliedVol

=S[twsuser]|tik!id[requestId]?askDelta

=S[twsuser]|tik!id[requestId]?lastImpliedVol

=S[twsuser]|tik!id[requestId]?lastDelta

=S[twsuser]|tik!id[requestId]?modelVolatility

=S[twsuser]|tik!id[requestId]?modelDelta

=S[twsuser]|tik!id[requestId]?modelPrice

=S[twsuser]|tik!id[requestId]?pvDividend

=S[twsuser]|tik!id[requestId]?modelGamma

=S[twsuser]|tik!id[requestId]?modelVega

=S[twsuser]|tik!id[requestId]?modelTheta

=S[twsuser]|tik!id[requestId]?modelUndPrice

5.5.3.6.2 Historical Data

In the example **Receiving the data** (p. **??**), you pulled midpoint historical data for EUR.USD from TWS.

Most of the formula's components are self-explanatory with the exception of duration and bar sizes, which require very specific codes as listed below:

5.5.3.6.2.1 Duration

Time Unit	Formula Abbreviation
Seconds	S
Day	D
Week	W
Month	M
Year	Y

5.5.3.6.2.2 Bar Sizes

Bar Size	Formula Parameter
1 second	1
5 seconds	2
15 seconds	3
30 seconds	4
1 minute	5
2 minutes	6
5 minutes	7
15 minutes	8
30 minutes	9
1 hour	10
1 day	11

For available **whatToShow** parameters and detailed descriptions, please refer to the **Historical Data Types** (p. **??**).

Please also be mindful of the **Historical Data Limitations** (p. **??**).

5.5.3.6.3 Contract Definitions

The TWS DDE for Excel API lets you retrieve data for any instrument available in TWS. So far we have been using the simplest instrument of all: CASH. Using slight variations of the same formula, you can define any security type available in TWS.

5.5.3.6.3.1 How to Find the Definition of a Contract

The best way of finding a contract's description is within the TWS itself. Within the TWS, you can easily check a contract's description either by double clicking it or through the **Contract Info ->** **Description** menu, which you access by right-clicking a contract in TWS:

The description will then appear:

5.5.3.6.3.2 Formulas for Different Security Types

Note: There are two different request types, i.e. "req1" and "req2", that differentiate in defining contracts with underlying Symbol (req1) or Local Symbol (req2) of the contract. While either request type can be used, the Local Symbol is most common for FUT, OPT and FOP.

Please find available contract parameters and sample contracts for each security type below:

FX Pairs

Formula

=S[twsuser]|tik!'id[reqId]?req?[symbol]_[SecType]_[exchange]_[currency]_~/'

Example

=Ssample123|tik!'id1?req?EUR_CASH_IDEALPRO_USD_~/'

STK

Formula

=S[twsuser]|tik!'id[reqId]?req?[symbol]_[SecType]_[exchange]_[currency]_~/'

Example

=Ssample123|tik!'id2?req?IBKR_STK_SMART_USD_~/'

FUT

FUT using the contract's local symbol

Formula

=S[twsuser]|tik!'id[reqId]?req2?[symbol]_[SecType]_[exchange]_[currency]_~/'

Example

=Ssample123|tik!'id3?req2?ESZ6_FUT_GLOBEX_USD_~/'

FUT using underlying's symbol, multiplier and expiration date

Formula

=S[twsuser]|tik!'id[reqId]?req?[underlying_symbol]_[SecType]_[expiry]_[multiplier]_[exchange]_[currency]_~_~/'

Example

=Ssample123|tik!'id3?req?ES_FUT_201612_50_GLOBEX_USD_~_~/'

OPT

OPT using the contract's local symbol

Formula

=S[twsuser]|tik!'id[reqId]?req2?[symbol]_[SecType]_[exchange]_[currency]_~/'

Example

=Ssample123|tik!'id4?req2?C DBK DEC 16 1300_OPT_DTB_EUR_~/'

Note: The format of the option local symbol conforms to the `Option Symbology Initiative (OSI).`

OPT using underlying's symbol, multiplier and expiration date

Formula

=S[twsuser]|tik!'id[reqId]?req?[underlying_symbol]_[SecType]_[expiry]_[strike]_[P/C]_[multiplier]_[exchange]_↩
[currency]_~_~/'

Example

=Ssample123|tik!'id4?req?DBK_OPT_20161216_13_C_100_DTB_EUR_∼_∼/'

FOP

FOP using the contract's local symbol

Formula

=S[twsuser]|tik!'id[reqId]?req2?[symbol]_[SecType]_[exchange]_[currency]_∼/'

Example

=Ssample123|tik!'id5?req2?XTZ6 C1100_FOP_GLOBEX_USD_∼/'

FOP using underlying's symbol, multiplier and expiration date

Formula

=S[twsuser]|tik!'id[reqId]?req?[underlying_symbol]_[SecType]_[expiry]_[strike]_[P/C]_[multiplier]_[exchange]_↩
[currency]_∼_[tradingClass]/'

Example

=Ssample123|tik!'id5?req?EUR_FOP_20161209_1.1_C_125000_GLOBEX_USD_∼_XT/'

IND

Formula

=S[twsuser]|tik!'id[reqId]?req?[symbol]_[SecType]_[exchange]_[currency]_∼/'

Example

=Ssample123|tik!'id6?req?ES_IND_GLOBEX_USD_∼/'

BAG

Formula

=S[twsuser]|tik!'id[reqId]?req?[symbol]_[SecType]_[exchange]_[currency]_CMBLGS_[num of legs]_[legId]_[leg↩
Quantity]_[legAction]_[legExchange]_[legPrice]...CMBLGS_∼/'

Example

=Ssample123|tik!'id7?req?

SPY_BAG_SMART_USD_CMBLGS_2_141149249_1_BUY_SMART_0_141149252_1_SELL_SMART_0_CMB↩
LGS_∼/'

5.6 ActiveX for Excel API

The ActiveX API wraps the C#/.NET API and is provided as an open source project TWSLib. It is suggested to also consider using the C# API directly as it provides seamless integration with the .NET framework.
One possible advantage of using the ActiveX for Excel API as compared to RTDServer or DDE is that ActiveX does provide the same number of functions as the other socket-based technologies (C#, Java, C++, Python). Disadvantages of the ActiveX Excel API is that it is more difficult to program as compared to other Excel APIs and not as robust as non-Excel socket-based API applications.

See **Excel API comparison**

5.6.1 ActiveX Sample Spreadsheet

A sample ActiveX for Excel spreadsheet is included with the API installation and installs to C:\TWS API\samples\↩ Excel\TwsActiveX.xls. The spreadsheet and the included ActiveX control are built for the 32 bit version of Excel for API versions until 973.05. With API versions 973.05+, the provided installed ActiveX control and Excel spreadsheet sample will work with either 32 bit or 64 bit applications. If version 9.73 of the ActiveX API is used, it is recommended to use version **973.07** or higher.

Important: Please note the sample ActiveX spreadsheet provided with the API is meant only as a demonstration of API functionality, and not intended as a production-level tool to be used in trading. While it is designed with examples of almost all API functions, it does not have the necessary functionality to handle problems that may occur during trading such as disconnections, error codes, or dropped events in a robust way.

Chapter 5

Troubleshooting & Support

http://interactivebrokers.github.io/tws-api/support.html

The API documentation contains a complete description of all API functions. Additionally the source code of the API itself is distributed freely and is a great resource for more in-depth understanding of how the API works. If after reviewing these resources there are remaining questions about available API functionality, the API Support group is available to help.

-> It is important to keep in mind that that IB **cannot provide programming assistance** or give suggestions on how to code custom applications. The API group can review log files which contain a record of communications between API applications and TWS, and give details about what the API can provide.

General suggestions on starting out with the IB system:

- **Become familiar with the analogous functionality in TWS before using the API**: the TWS API is nothing but a communication channel between your client application and TWS. Each API function has a corresponding tool in TWS. For instance, the market data tick types in the API correspond to watchlist columns in TWS. Any order which can be created in the API can first be created in TWS, and it is recommended to do so. Additionally, if information is not available in TWS, it will not be available in the API. Before using IB Gateway with the API, it is recommended to first become familiar with TWS.

- **Make use of the sample API applications**: the sample applications distributed with the API download have examples of essentially every API function in each of the available programming languages. If an issue does not occur in the corresponding sample application, that implies there is a problem with the custom implementation.

- **Upgrade TWS or IB Gateway periodically**: TWS and IB Gateway often have new software releases that have enhancements, and that can sometimes have bug fixes. Because of this, we strongly recommend our users to keep their software as up to date as possible. There is no problem with staying with a version of the API and upgrading TWS, as TWS/IB Gateway are designed to be backwards compatible with older API versions. If you are experiencing a specific problem that is occurring in TWS or IB Gateway and not in the API program, it is quite possible it does not occur in the more recent software build.

6.1 Log Files

The log files are essential to provide detailed information about how a custom application may be malfunctioning. They are useful tools for direct review by an API application programmer, and additionally they can be uploaded for review by the API Support group.

6.1.1 API Log

TWS and IB Gateway can be configured to create a separate log file which has a record of just communications with API applications. This log is not enabled by default; but needs to be enabled by the Global Configuration setting **"Create API Message Log File"**(picture below).

- API logs contain a record of exchanged messages between API applications and TWS/IB Gateway. Since only API messages are recorded, the API logs are more compact and easier to handle. However they do not contain general diagnostic information about TWS/IBG as the TWS/IBG logs. In TWS/IBG versions above 956, the API message log files will be created within a subdirectory of the TWS settings folder that is specific to that user. The TWS/IBG settings folder is by default **C:\Jts** (or IBJts on Mac/Linux). The API logs are named **api.[clientId].[day].log**, where [clientId] corresponds to the Id the client application used to connect to the TWS and [day] to the week day (i.e. api.123.Thu.log).
- There is also a setting "Include Market Data in API Log" that will include streaming market data values in the API log file.

Enabling creation of API logs

TWS:

- Navigate to File/Edit → Global Configuration → API → Settings
- Check the box *Create API message log file*
- Click Apply and Ok

IB Gateway:

- Navigate to Configure → Settings → API → Settings
- Check the box *Create API message log file*
- Click Apply and Ok

6.1.2 TWS Log File

The TWS Logging Level must be set to the 'Detail' level to record information pertinent to the API. By default it is at the 'Error' level which records a minimum of diagnostic information. To capture API messages it is necessary to change the Logging Level to 'Detail'. Note the 'Logging Level', like all TWS/IBG settings, is saved separately for different users and for TWS and IBG.

- **Important:** The TWS/IB Gateway log file setting has to be set to 'Detail' level before an issue occurs so that information recorded correctly when it manifests. However due to the high amount of information that will be generated under this level, the resulting logs can grow considerably in size. It is therefore recommended to use the 'Detail' level only when troubleshooting and/or tracking a very specific issue. This can also be done from the API client itself using the **IBApi.EClient.setServerLogLevel** function. Some third party applications, such as NinjaTrader, are configured to invoke this function to set the TWS Logging Level every time they connect, and so to set the TWS Log to 'Detail' this will have to be done from within the API client program.

Steps to set TWS Logging Level to Detail

TWS:

- Navigate to File/Edit → Global Configuration → API → Settings
- Set *Logging Level* to *Detail*

IB Gateway:

- Navigate to Configure → Settings → API → Settings
- Set *Logging Level* to *Detail*

6.1.3 Local location of logs

For TWS and IB Gateway versions above 956 the TWS, IB Gateway, and API logs are stored in a common folder for each username. This folder is within the TWS settings directory, C:\Jts\ by default on a Windows computer (the default can be configured differently on the login screen).

The path to the log file directory can be found from a TWS or IB Gateway session by using the combination **Ctrl-Alt-U**. This will reveal path such as C:\Jts\detcfsvirl\ (on Windows).

6.1.4 Uploading logs

- If API logging has been enabled with the setting "Create API Message Log" during the time when an issue occurs, it can be uploaded to the API group using the combination **Ctrl-Alt-Q**, then clicking **Submit**. If logs have been uploaded, please let the API Support group know by **creating a webticket** in the Message Center in Account Management (under Support) indicating the **username** of the associated TWS session. In some cases a TWS log may also be requested at the Detailed logging level. The TWS log can grow quite large and may not be uploadable by the automatic method; in this case an alternative means of upload can be found.

Chapter 6

Programming the API: Architecture

http://interactivebrokers.github.io/tws-api/client_wrapper.html

7.1 EClientSocket and EWrapper Classes

Once the TWS is up and running and actively listening for incoming connections we are ready to write our code. This brings us to the TWS API's two major classes: the **IBApi.EWrapper** interface and the **IBApi.EClientSocket**

7.2 Implementing the EWrapper Interface

The **IBApi.EWrapper** interface is the mechanism through which the TWS delivers information to the A↩PI client application. By implementing this interface the client application will be able to receive and handle the information coming from the TWS. For further information on how to implement interfaces, refer to your programming language's documentation.

- ```
  public class EWrapperImpl : EWrapper
  {
  ```
- ```
  public class EWrapperImpl implements EWrapper {
  ```
- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```
- ```
  class TestCppClient : public EWrapper
  {
  ```
- ```
  class TestWrapper(wrapper.EWrapper):
  ```

7.3 The EClientSocket Class

The class used to send messages to TWS is **IBApi.EClientSocket** (p. **??**). Unlike EWrapper, this class is not overriden as the provided functions in EClientSocket are invoked to send messages to TWS. To use EClientSocket, first it may be necessary to implement the **IBApi.EWrapper** interface as part of its constructor parameters so that the application can handle all returned messages. Messages sent from TWS as a response to function calls in **IBApi.EClientSocket** require a EWrapper implementation so they can processed to meet the needs of the API client.

Another crucial element is the **IBApi.EReaderSignal** object passed to theEClientSocket's constructor. With the exception of Python, this object is used in APIs to signal a message is ready for processing in the queue. (In Python the Queue class handles this task directly). We will discuss this object in more detail in the **The EReader Thread** section.
  ```

- ```
        EClientSocket clientSocket;
        public readonly EReaderSignal Signal;
  ```

  ...

  ```
        public EWrapperImpl()
        {
            Signal = new EReaderMonitorSignal();
            clientSocket = new EClientSocket(this, Signal);
        }
  ```

- ```
    private EReaderSignal readerSignal;
    private EClientSocket clientSocket;
    protected int currentOrderId = -1;
  ```

  ...

  ```
    public EWrapperImpl() {
        readerSignal = new EJavaSignal();
        clientSocket = new EClientSocket(this, readerSignal);
    }
  ```

- ```
        Public eReaderSignal As EReaderSignal
        Public socketClient As EClientSocket
  ```

  ...

  ```
        Sub New()
            eReaderSignal = New EReaderMonitorSignal
            socketClient = New EClientSocket(Me, eReaderSignal)
        End Sub
  ```

- ```
    EReaderOSSignal m_osSignal;
    EClientSocket * const m_pClient;
  ```

  ...

  ```
  TestCppClient::TestCppClient() :
        m_osSignal(2000)//2-seconds timeout
    , m_pClient(new EClientSocket(this, &m_osSignal))
    , m_state(ST_CONNECT)
    , m_sleepDeadline(0)
    , m_orderId(0)
    , m_pReader(0)
    , m_extraAuth(false)
  {
  }
  ```

- ```
  class TestClient(EClient):
      def __init__(self, wrapper):
          EClient.__init__(self, wrapper)
  ```

  ...

  ```
  class TestApp(TestWrapper, TestClient):
      def __init__(self):
          TestWrapper.__init__(self)
          TestClient.__init__(self, wrapper=self)
  ```

Note: *The EReaderSignal class is not used for Python API. The Python Queue module is used for inter-thread communication and data exchange.*
  ```

Chapter 7

Connectivity http://interactivebrokers.github.io/tws-api/connection.html

A TCP connection between the API client application and TWS needs to first be established via the **IBApi.EClient↩ Socket.eConnect** function. TWS acts as a server to receive requests from the API application (the client) and responds by taking appropriate actions. The first step is for the API client to initiate a connection to the socket port on which TWS is listening. It is possible to have multiple TWS instances running on the same computer if each is configured with a different API socket port number. Also, each TWS session can receive up to **32 different client applications** simultaneously. The **client ID** field specified in the API connection is used to distinguish different API clients.

8.1 Establishing an API connection

Once our two main objects have been created, EWrapper and ESocketClient, the client application can connect via the **IBApi.EClientSocket** object:

- `clientSocket.eConnect("127.0.0.1", 7497, 0);`
- `m_client.eConnect("127.0.0.1", 7497, 2);`
- `socketClient.eConnect("127.0.0.1", 7497, 0)`
- `bool bRes = m_pClient->eConnect( host, port, clientId, m_extraAuth);`
- `app.connect("127.0.0.1", args.port, clientId=0)`

eConnect starts by requesting from the OS that a TCP socket be opened to the specified IP address and host number- the first two parameters in the function call. If the socket cannot be opened, the OS returns an error condition which the API returns as error code 502 to **IBApi.EWrapper.error** (p. **??**). Since this error is not generated by TWS it is not captured in TWS log files. Most commonly error 502 will indicate that TWS is not running with the API enabled or is listening for connection requests on a different socket port. If connecting across a network, it can also occur if there is a firewall or antivirus program blocking connections.

After the socket has been opened, both applications need to exchange essential information for the API session. First, the version numbers of the client (API program) and server (TWS) are exchanged so that the server and client each know the function calls supported by one another. In this way backwards compatibility can be maintained between TWS and previous API versions. Next TWS will always return certain information for the client session, namely the accounts which are accessible by the TWS session, the next valid order identifier (ID), and the time of connection. In the most common mode of operation the EClient.AsyncEConnect field is set to false and the initial handshake is taken to completion immediately after the socket connection is established. TWS will then immediately provides the API client with this information.

- Important: The **IBApi.EWrapper.nextValidID** callback is commonly used to indicate that the connection is completed and other messages can be sent from the API client to TWS. There is the possibility that function calls made prior to this time could be dropped by TWS.

There is also an alternative mode of connection used in special cases in wich the variable AsyncEconnect is set to true, and the call to startAPI is only called from the connectAck() function. All IB samples use the mode Async↩ Econnect = False.

8.2 The EReader Thread

API programs always have at least two threads of execution. One thread is used for sending messages to TWS, and another thread is used for reading returned messages. The second thread uses the API EReader class to read from the socket and add messages to a queue. Everytime a new message is added to the message queue, a notification flag is triggered to let other threads now that there is a message waiting to be processed. In the two-thread design of an API program, the message queue is also processed by the first thread. In a three-thread design, an additional thread is created to perform this task. The thread responsible for the message queue will decode messages and invoke the appropriate functions in EWrapper. The two-threaded design is used in the IB Python sample Program.py and the C++ sample TestCppClient, while the 'Testbed' samples in the other languages use a three-threaded design. Commonly in a Python asynchronous network application, the `asyncio module` will be used to create a more sequential looking code design.

The class which has functionality for reading and parsing raw messages from TWS is the **IBApi.EReader** class.

- ```
            //Create a reader to consume messages from the TWS. The EReader will consume the incoming
       messages and put them in a queue
            var reader = new EReader(clientSocket, readerSignal);
            reader.Start();
            //Once the messages are in the queue, an additional thread can be created to fetch them
            new Thread(() => { while (clientSocket.IsConnected()) { readerSignal.waitForSignal(); reader.
      processMsgs(); } }) { IsBackground = true }.Start();
  ```

- ```
          final EReader reader = new EReader(m_client, m_signal);

          reader.start();
          //An additional thread is created in this program design to empty the messaging queue
          new Thread(() -> {
              while (m_client.isConnected()) {
                  m_signal.waitForSignal();
                  try {
                      reader.processMsgs();
                  } catch (Exception e) {
                      System.out.println("Exception: "+e.getMessage());
                  }
              }
          }).start();
  ```

- ```
          'Once the messages are in the queue, an additional thread need to fetch them
          Dim msgThread As Thread = New Thread(AddressOf messageProcessing)
          msgThread.IsBackground = True
          If (wrapperImpl.serverVersion() > 0) Then Call msgThread.Start()
  ```

  ...

  ```
      Private Sub messageProcessing()
          Dim reader As EReader = New EReader(wrapperImpl.socketClient, wrapperImpl.eReaderSignal)
          reader.Start()
          While (wrapperImpl.socketClient.IsConnected)
              wrapperImpl.eReaderSignal.waitForSignal()
              reader.processMsgs()
          End While
      End Sub
  ```

- ```
          m_pReader = new EReader(m_pClient, &m_osSignal);
          m_pReader->start();
  ```

- In Python IB API, the code below is included in Client::connect(), so the EReader thread is automatically started upon connection. There is **no need** for user to start the reader.

  ```
          # You don't need to run this in your code!
          self.reader = reader.EReader(self.conn, self.msg_queue)
          self.reader.start()   # start thread
  ```

  Once the client is connected, a reader thread will be automatically created to handle incoming messages and put the messages into a message queue for further process. User **is required** to trigger Client::run() below, where the message queue is processed in an infinite loop and the EWrapper call-back functions are automatically triggered.

```
app.run()
```

Now it is time to revisit the role of **IBApi.EReaderSignal** initially introduced in **The EClientSocket Class** (p. **??**). As mentioned in the previous paragraph, after the EReader thread places a message in the queue, a notification is issued to make known that a message is ready for processing. In the (C++, C#/.NET, Java) APIs, this is done via the **IBApi.EReaderSignal** object we initiated within the **IBApi.EWrapper** (p. **??**)'s implementer. In the Python API, it is handled automatically by the `Queue class`.

The client application is now ready to work with the Trader Workstation! At the completion of the connection, the API program will start receiving events such as **IBApi.EWrapper.nextValidId** and **IBApi.EWrapper.managed↩ Accounts** (p. **??**). In TWS (*not IB Gateway*) if there is an active network connection, there will also immediately be callbacks to **IBApi::EWrapper::error** with errorId as -1 and errorCode=*2104*,*2106*, errorMsg = "Market Data Server is ok" to indicate there is an active connection to the IB market data server. Callbacks to **IBApi::E↩Wrapper::error** with errorId as -1 do not represent true 'errors' but only notifications that a connection has been made successfully to the IB market data farms.

IB Gateway by contrast will not make connections to market data farms until a request is made by the IB client. Until this time the connection indicator in the IB Gateway GUI will show a yellow color of 'inactive' rather than an 'active' green indication.

When initially making requests from an API application it is important that the verifies that a response is received rather than proceeding assuming that the network connection is ok and the subscription request (portfolio updates, account information, etc) was made successfully.

8.3 Accepting an API connection from TWS

For security reasons, by default the API is not configured to automatically accept connection requests from API applications. After a connection attempt, a dialogue will appear in TWS asking the user to manually confirm that a connection can be made:

To prevent the TWS from asking the end user to accept the connection, it is possible to configure it to automatically accept the connection from a trusted IP address and/or the local machine. This can easily be done via the TWS API settings:

Note: you have to make sure the connection has been fully established before attempting to do any requests to the TWS. Failure to do so will result in the TWS closing the connection. Typically this can be done by waiting for a callback from an event and the end of the initial connection handshake, such as **IBApi.EWrapper.nextValidId** or **IBApi.EWrapper.managedAccounts** (p. **??**).

In rare cases in which IB Gateway or TWS has a momentarily delay in establishing connecting to the IB servers, messages sent immediately after receiving the nextValidId could be dropped and would need to be resent. If the API client has not receive the expected callbacks from issued requests, it should not proceed assumming the connection is ok.

8.4 Broken API socket connection

If there is a problem with the socket connection between TWS and the API client, for instance if TWS suddenly closes, this will trigger an exception in the EReader thread which is reading from the socket. This exception will also occur if an API client attempts to connect with a client ID that is already in use.

The socket EOF is handled slightly differently in different API languages. For instance in Java, it is caught and sent to the client application to **IBApi::EWrapper::error** with errorCode 507: "Bad Message". In C# it is caught and sent to **IBApi::EWrapper::error** with errorCode -1. The client application needs to handle this error message and use it to indicate that an exception has been thrown in the socket connection. Associated functions such as **IBApi::EWrapper::connectionClosed** and **IBApi::EClient::IsConnected** functions are not called automatically by the API code but need to be handled at the API client-level∗.

- This has been changed in API version 973.04

Chapter 8

Financial Instruments (Contracts)

http://interactivebrokers.github.io/tws-api/contracts.html

9.1 Overview

An **IBApi.Contract** object represents trading instruments such as a stocks, futures or options.

Every time a new request that requires a contract (i.e. market data, order placing, etc.) is sent to TWS, the platform will try to match the provided contract object with a single candidate. If there is more than one contract matching the same description, TWS will return an error notifying you there is an ambiguity. In these cases the TWS needs further information to narrow down the list of contracts matching the provided description to a single element.

The best way of finding a contract's description is within the TWS itself. Within the TWS, you can easily check a contract's description either by double clicking it or through the **Contract Info ->** **Description** menu, which you access by right-clicking a contract in TWS:

The description will then appear:

Note: you can see the extended contract details by choosing **Contract Info ->** **Details**. This option will open a web page showing all available information on the contract.

Whenever a contract description is provided via the TWS API, the TWS will try to match the given description to a single contract. This mechanism allows for great flexibility since it gives the possibility to define the same contract in multiple ways.

The simplest way to define a contract is by providing its symbol, security type, currency and exchange. The vast majority of stocks, CFDs, Indexes or FX pairs can be uniquely defined through these four attributes. More complex contracts such as options and futures require some extra information due to their nature. Below are several examples for different types of instruments.

See also:

- **Requesting Contract Details**
- **Stock Contract Search**
- **Basic Contracts**
- **Spreads**

9.2 Requesting Contract Details

Complete details about a contract in IB's database can be retrieved using the function **IBApi.EClient.reqContract↩ Details** (p. **??**). This includes information about a contracts conID, symbol, local symbol, currency, etc. which is returned in a **IBApi.ContractDetails** object IBApi.EWrapper.contractDetail. reqContractDetails takes as an argument a Contract object which may uniquely match an argument, and unlike other API functions it can also take a Contract object which matches multiple contracts in IB's database. When there are multiple matches, they will each be returned individually to the function **IBApi::EWrapper::contractDetails** (p. **??**).

Note: Invoking reqContractDetails with a Contract object which has currency = USD will only return US contracts, even if there are non-US instruments which have the USD currency.

Request for Bond details will be returned to **IBApi::EWrapper::bondContractDetails** instead. Because of bond market data license restrictions, there are only a few available fields to be returned in a bond contract description, namely the minTick, exchange, and short name.

One particular use of the **IBApi::EClient::reqContractDetails** function is to request an option chain. See **Option Chains** for more details.

9.2.1 BroadTape News List

The example below shows an "incomplete" news **IBApi.Contract** with no symbol or currency defined. In most cases using such a contract would result in an invalid contract details error since a symbol or localSymbol is required. **IBApi.EClient.reqContractDetails** will instead use it to obtain the whole BroadTape news chain from the TWS.

-
```
            Contract contract = new Contract();
            contract.SecType = "NEWS";
            contract.Exchange = "BT"; //Briefing Trader
```
...
```
            client.reqContractDetails(211, ContractSamples.NewsFeedForQuery());
```

-
```
        Contract contract = new Contract();
        contract.secType("NEWS");
        contract.exchange("BT"); //Briefing Trader
```
...
```
        client.reqContractDetails(211, ContractSamples.NewsFeedForQuery());
```

-
```
            Dim contract As Contract = New Contract()
            contract.SecType = "NEWS"
            contract.Exchange = "BT" 'Briefing Trader
```
...
```
        client.reqContractDetails(211, ContractSamples.NewsFeedForQuery())
```

-
```
    Contract contract;
    contract.secType = "NEWS";
    contract.exchange = "BT"; //Briefing Trader
```
...
```
    m_pClient->reqContractDetails(211, ContractSamples::NewsFeedForQuery());
```

-
```
        contract = Contract()
        contract.secType = "NEWS"
        contract.exchange = "BT" #Briefing Trader
```
...

```
        self.reqContractDetails(213, ContractSamples.NewsFeedForQuery())
```

All returned objects will be delivered via **IBApi.EWrapper.contractDetails** (p. **??**). Once all contracts have been delivered the **IBApi.EWrapper.contractDetailsEnd** marker will be triggered to notify it.

- ```
    public class EWrapperImpl : EWrapper
    {
  ```

  ...

  ```
        public virtual void contractDetails(int reqId, ContractDetails contractDetails)
        {
            Console.WriteLine("ContractDetails begin. ReqId: " + reqId);
            printContractMsg(contractDetails.Contract);
            printContractDetailsMsg(contractDetails);
            Console.WriteLine("ContractDetails end. ReqId: " + reqId);
        }
  ```

  ...

  ```
        public virtual void contractDetailsEnd(int reqId)
        {
            Console.WriteLine("ContractDetailsEnd. "+reqId+"\n");
        }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void contractDetails(int reqId, ContractDetails contractDetails) {
          System.out.println(EWrapperMsgGenerator.contractDetails(reqId, contractDetails));
      }
  ```

  ...

  ```
      @Override
      public void contractDetailsEnd(int reqId) {
          System.out.println("ContractDetailsEnd. "+reqId+"\n");
      }
  ```

- ```
    Public Class EWrapperImpl
        Implements EWrapper
  ```

  ...

  ```
        Public Sub contractDetails(reqId As Integer, contractDetails As IBApi.ContractDetails) Implements
       IBApi.EWrapper.contractDetails
            Console.WriteLine("ContractDetails begin. ReqId: " & reqId)
            printContractMsg(contractDetails.Contract)
            printContractDetailsMsg(contractDetails)
            Console.WriteLine("ContractDetails end. ReqId: " & reqId)
        End Sub
  ```

  ...

  ```
        Public Sub contractDetailsEnd(reqId As Integer) Implements IBApi.EWrapper.contractDetailsEnd
            Console.WriteLine("ContractDetailsEnd - ReqId [" & reqId & "]")
        End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::contractDetails( int reqId, const ContractDetails& contractDetails) {
      printf( "ContractDetails begin. ReqId: %d\n", reqId);
      printContractMsg(contractDetails.contract);
      printContractDetailsMsg(contractDetails);
      printf( "ContractDetails end. ReqId: %d\n", reqId);
  }
  ```

...

```
void TestCppClient::contractDetailsEnd( int reqId) {
    printf( "ContractDetailsEnd. %d\n", reqId);
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

...

```
    def contractDetails(self, reqId: int, contractDetails: ContractDetails):
        super().contractDetails(reqId, contractDetails)
        printinstance(contractDetails.contract)
```

...

```
    def contractDetailsEnd(self, reqId: int):
        super().contractDetailsEnd(reqId)
        print("ContractDetailsEnd. ", reqId, "\n")
```

Important: due to the potentially high amount of data resulting from such queries this request is subject to pacing. Although a request such as the above one will be answered immediately, a similar subsequent one will be kept on hold for one minute. This amount of time will increase if more such requests are performed. To prevent this narrow down the amount of eligible contracts by providing an expiration date specifying at least the year (i.e. 2016) or the year and the month (i.e. 201603 for March 2016).

9.3 Stock Contract Search

Starting in API **v973.02** and TWS **v964**, a function **IBApi::EClient::reqMatchingSymbols** is available to search for stock contracts. The input can be either the first few letters of the ticker symbol, or for longer strings, a character sequence matching a word in the security name. For instance to search for the stock symbol 'IBKR', the input 'I' or 'IB' can be used, as well as the word 'Interactive'. Up to 16 matching results are returned.

- There must be an interval of at least 1 second between successive calls to reqMatchingSymbols

- ```
              client.reqMatchingSymbols(211, "IB");
  ```
- ```
          client.reqMatchingSymbols(211, "IB");
  ```
- ```
          client.reqMatchingSymbols(202, "IB")
  ```
- ```
      m_pClient->reqMatchingSymbols(11001, "IBM");
  ```
- ```
          self.reqMatchingSymbols(212, "IB")
  ```

Matching stock contracts are returned to **IBApi::EWrapper::symbolSamples** with information about types of derivative contracts which exist (warrants, options, dutch warrants, futures).

-
```
        public void symbolSamples(int reqId, ContractDescription[] contractDescriptions)
        {
            string derivSecTypes;
            Console.WriteLine("Symbol Samples. Request Id: {0}", reqId);

            foreach (var contractDescription in contractDescriptions)
            {
                derivSecTypes = "";
                foreach (var derivSecType in contractDescription.DerivativeSecTypes)
                {
                    derivSecTypes += derivSecType;
                    derivSecTypes += " ";
                }
                Console.WriteLine("Contract: conId - {0}, symbol - {1}, secType - {2}, primExchange - {3},
   currency - {4}, derivativeSecTypes - {5}",
                    contractDescription.Contract.ConId, contractDescription.Contract.Symbol,
  contractDescription.Contract.SecType,
                    contractDescription.Contract.PrimaryExch, contractDescription.Contract.Currency,
  derivSecTypes);
            }
        }
```

-
```
    @Override
    public void symbolSamples(int reqId, ContractDescription[] contractDescriptions) {
        System.out.println("Contract Descriptions. Request: " + reqId + "\n");
        for (ContractDescription  cd : contractDescriptions) {
            Contract c = cd.contract();
            StringBuilder derivativeSecTypesSB = new StringBuilder();
            for (String str : cd.derivativeSecTypes()) {
                derivativeSecTypesSB.append(str);
                derivativeSecTypesSB.append(",");
            }
            System.out.print("Contract. ConId: " + c.conid() + ", Symbol: " + c.symbol() + ", SecType: " +
  c.secType() +
                    ", PrimaryExch: " + c.primaryExch() + ", Currency: " + c.currency() +
                    ", DerivativeSecTypes:[" + derivativeSecTypesSB.toString() + "]");
        }

        System.out.println();
    }
```

-
```
        Public Sub symbolSamples(reqId As Integer, contractDescriptions As ContractDescription())
       Implements EWrapper.symbolSamples
            Dim derivSecTypes As String

            Console.WriteLine("Symbol Samples. Request Id: " & reqId)

            For Each contractDescription In contractDescriptions
                derivSecTypes = ""
                For Each derivSecType In contractDescription.DerivativeSecTypes
                    derivSecTypes += derivSecType
                    derivSecTypes += " "
                Next
                Console.WriteLine("Contract: conId - " & contractDescription.Contract.ConId & ", symbol - "
   & contractDescription.Contract.Symbol &
                                  ", secType -" & contractDescription.Contract.SecType & ", primExchange -
   " & contractDescription.Contract.PrimaryExch &
                                  ", currency - " & contractDescription.Contract.Currency & ",
   derivativeSecTypes - " & derivSecTypes)
            Next
        End Sub
```

-
```
void TestCppClient::symbolSamples(int reqId, const std::vector<ContractDescription> &contractDescriptions)
      {
    printf("Symbol Samples (total=%lu) reqId: %d\n", contractDescriptions.size(), reqId);

    for (unsigned int i = 0; i < contractDescriptions.size(); i++) {
        Contract contract = contractDescriptions[i].contract;
        std::vector<std::string> derivativeSecTypes = contractDescriptions[i].derivativeSecTypes;
        printf("Contract (%u): %ld %s %s %s %s, ", i, contract.conId, contract.symbol.c_str(), contract.
  secType.c_str(), contract.primaryExchange.c_str(), contract.currency.c_str());
        printf("Derivative Sec-types (%lu):", derivativeSecTypes.size());
        for (unsigned int j = 0; j < derivativeSecTypes.size(); j++) {
            printf(" %s", derivativeSecTypes[j].c_str());
        }
        printf("\n");
    }
}
```

-
```
    def symbolSamples(self, reqId: int,
                      contractDescriptions: ListOfContractDescription):
        super().symbolSamples(reqId, contractDescriptions)
        print("Symbol Samples. Request Id: ", reqId)

        for contractDescription in contractDescriptions:
            derivSecTypes = ""
            for derivSecType in contractDescription.derivativeSecTypes:
```

```
                derivSecTypes += derivSecType
                derivSecTypes += " "
            print("Contract: conId:%s, symbol:%s, secType:%s primExchange:%s, "
                  "currency:%s, derivativeSecTypes:%s" % (
                contractDescription.contract.conId,
                contractDescription.contract.symbol,
                contractDescription.contract.secType,
                contractDescription.contract.primaryExchange,
                contractDescription.contract.currency, derivSecTypes))
```

9.4 Basic Contracts

9.4.1 FX Pairs

-
```
            Contract contract = new Contract();
            contract.Symbol = "EUR";
            contract.SecType = "CASH";
            contract.Currency = "GBP";
            contract.Exchange = "IDEALPRO";
```

-
```
        Contract contract = new Contract();
        contract.symbol("EUR");
        contract.secType("CASH");
        contract.currency("GBP");
        contract.exchange("IDEALPRO");
```

-
```
            Dim contract As Contract = New Contract
            contract.Symbol = "EUR"
            contract.SecType = "CASH"
            contract.Currency = "GBP"
            contract.Exchange = "IDEALPRO"
```

-
```
    Contract contract;
    contract.symbol = "EUR";
    contract.secType = "CASH";
    contract.currency = "GBP";
    contract.exchange = "IDEALPRO";
```

-
```
        contract = Contract()
        contract.symbol = "EUR"
        contract.secType = "CASH"
        contract.currency = "GBP"
        contract.exchange = "IDEALPRO"
```

9.4.2 Stocks

-
```
            Contract contract = new Contract();
            contract.Symbol = "IBKR";
            contract.SecType = "STK";
            contract.Currency = "USD";
            //In the API side, NASDAQ is always defined as ISLAND in the exchange field
            contract.Exchange = "ISLAND";
```

-
```
        Contract contract = new Contract();
        contract.conid(46636665);
        contract.secType("STK");
        //contract.currency("USD");
        //In the API side, NASDAQ is always defined as ISLAND
        contract.exchange("SEHK");
```

-
```
            Dim contract As Contract = New Contract
            contract.Symbol = "IBKR"
            contract.SecType = "STK"
            contract.Currency = "USD"
            '! In the API side, NASDAQ Is always defined as ISLAND for routing purposes
            contract.Exchange = "ISLAND"
```

```
Contract contract;
contract.symbol = "IBKR";
contract.secType = "STK";
contract.currency = "USD";
//In the API side, NASDAQ is always defined as ISLAND
contract.exchange = "ISLAND";
```

```
contract = Contract()
contract.symbol = "IBKR"
contract.secType = "STK"
contract.currency = "USD"
#In the API side, NASDAQ is always defined as ISLAND in the exchange field
contract.exchange = "ISLAND"
```

For certain smart-routed stock contracts that have the same **symbol**, **currency** and **exchange**, you would also need to specify the **primary exchange** attribute to uniquely define the contract. This should be defined as the native exchange of a contract, and is good practice to include for all stocks:

```
Contract contract = new Contract();
contract.Symbol = "MSFT";
contract.SecType = "STK";
contract.Currency = "USD";
contract.Exchange = "SMART";
//Specify the Primary Exchange attribute to avoid contract ambiguity
// (there is an ambiguity because there is also a MSFT contract with primary exchange = "AEB")
contract.PrimaryExch = "ISLAND";
```

```
Contract contract = new Contract();
contract.symbol("MSFT");
contract.secType("STK");
contract.currency("USD");
contract.exchange("SMART");
// Specify the Primary Exchange attribute to avoid contract ambiguity
// (there is an ambiguity because there is also a MSFT contract with primary exchange = "AEB")
contract.primaryExch("ISLAND");
```

```
Dim Contract As Contract = New Contract
Contract.symbol = "MSFT"
Contract.secType = "STK"
Contract.currency = "USD"
Contract.exchange = "SMART"
'Specify the Primary Exchange attribute to avoid contract ambiguity
'(there is an ambiguity because there is also a MSFT contract with primary exchange = "AEB")
Contract.PrimaryExch = "ISLAND"
```

```
Contract contract;
contract.symbol = "AAPL";
contract.secType = "STK";
contract.currency = "USD";
contract.exchange = "SMART";
// Specify the Primary Exchange attribute to avoid contract ambiguity
// (there is an ambiguity because there is also a MSFT contract with primary exchange = "AEB")
contract.primaryExchange = "ISLAND";
```

```
contract = Contract()
contract.symbol = "MSFT"
contract.secType = "STK"
contract.currency = "USD"
contract.exchange = "SMART"
#Specify the Primary Exchange attribute to avoid contract ambiguity
#(there is an ambiguity because there is also a MSFT contract with primary exchange = "AEB")
contract.primaryExchange = "ISLAND"
```

For the purpose of requesting market data, the routing exchange and primary exchange can be specified in a single 'exchange' field if they are separated by a valid component exchange separator, for instance exchange = "SMAR↩ T:ARCA". The default separators available are colon ":" and slash "/". Other component exchange separators can be defined using the field defined in TWS Global Configuration under API -> Settings. The component exchange separator syntax in TWS versions prior to 971 can only be used to request market data and not to place orders.

9.4.3 Indexes

ISINs for indices which are available in IB's database are available in the API as of TWS 965+.

- ```
            Contract contract = new Contract();
            contract.Symbol = "DAX";
            contract.SecType = "IND";
            contract.Currency = "EUR";
            contract.Exchange = "DTB";
  ```

- ```
        Contract contract = new Contract();
        contract.symbol("DAX");
        contract.secType("IND");
        contract.currency("EUR");
        contract.exchange("DTB");
  ```

- ```
            Dim contract As Contract = New Contract
            contract.Symbol = "DAX"
            contract.SecType = "IND"
            contract.Currency = "EUR"
            contract.Exchange = "DTB"
  ```

- ```
    Contract contract;
    contract.symbol = "DAX";
    contract.secType = "IND";
    contract.currency = "EUR";
    contract.exchange = "DTB";
  ```

- ```
        contract = Contract()
        contract.symbol = "DAX"
        contract.secType = "IND"
        contract.currency = "EUR"
        contract.exchange = "DTB"
  ```

9.4.4 CFDs

- ```
            Contract contract = new Contract();
            contract.Symbol = "IBDE30";
            contract.SecType = "CFD";
            contract.Currency = "EUR";
            contract.Exchange = "SMART";
  ```

- ```
        Contract contract = new Contract();
        contract.symbol("IBDE30");
        contract.secType("CFD");
        contract.currency("EUR");
        contract.exchange("SMART");
  ```

- ```
            Dim contract As Contract = New Contract
            contract.Symbol = "IBDE30"
            contract.SecType = "CFD"
            contract.Currency = "EUR"
            contract.Exchange = "SMART"
  ```

- ```
    Contract contract;
    contract.symbol = "IBDE30";
    contract.secType = "CFD";
    contract.currency = "EUR";
    contract.exchange = "SMART";
  ```

- ```
        contract = Contract()
        contract.symbol = "IBDE30"
        contract.secType = "CFD"
        contract.currency = "EUR"
        contract.exchange = "SMART"
  ```

9.4.5 Futures

A regular futures contract is commonly defined using an expiry and the symbol field defined as the symbol of the underlying. Historical data for futures is available up to 2 years after they expire by setting the includeExpired flag within the Contract class to True.

- ```
            Contract contract = new Contract();
            contract.Symbol = "ES";
            contract.SecType = "FUT";
            contract.Exchange = "GLOBEX";
            contract.Currency = "USD";
            contract.LastTradeDateOrContractMonth = "201803";
  ```
  ```

```
Contract contract = new Contract();
contract.symbol("ES");
contract.secType("FUT");
contract.currency("USD");
contract.exchange("GLOBEX");
contract.lastTradeDateOrContractMonth("201803");
```

```
    Dim contract As Contract = New Contract
    contract.Symbol = "ES"
    contract.SecType = "FUT"
    contract.Exchange = "GLOBEX"
    contract.Currency = "USD"
    contract.LastTradeDateOrContractMonth = "201803"
```

```
Contract contract;
contract.symbol = "ES";
contract.secType = "FUT";
contract.exchange = "GLOBEX";
contract.currency = "USD";
contract.lastTradeDateOrContractMonth = "201803";
```

```
contract = Contract()
contract.symbol = "ES"
contract.secType = "FUT"
contract.exchange = "GLOBEX"
contract.currency = "USD"
contract.lastTradeDateOrContractMonth = "201803"
```

By contract the 'local symbol' field is IB's symbol for the future itself (the Symbol within the TWS' Contract Description dialog). Since a local symbol uniquely defines a future, an expiry is not necessary.

```
    Contract contract = new Contract();
    contract.SecType = "FUT";
    contract.Exchange = "GLOBEX";
    contract.Currency = "USD";
    contract.LocalSymbol = "ESU6";
```

```
Contract contract = new Contract();
contract.localSymbol("ESU6");
contract.secType("FUT");
contract.currency("USD");
contract.exchange("GLOBEX");
```

```
    Dim contract As Contract = New Contract
    contract.SecType = "FUT"
    contract.Exchange = "GLOBEX"
    contract.Currency = "USD"
    contract.LocalSymbol = "ESU6"
```

```
Contract contract;
contract.secType = "FUT";
contract.exchange = "GLOBEX";
contract.currency = "USD";
contract.localSymbol = "ESZ6";
```

```
contract = Contract()
contract.secType = "FUT"
contract.exchange = "GLOBEX"
contract.currency = "USD"
contract.localSymbol = "ESU6"
```

Occasionally, you can expect to have more than a single future contract for the same underlying with the same expiry. To rule out the ambiguity, the contract's **multiplier** can be given as shown below:

```
    Contract contract = new Contract();
    contract.Symbol = "DAX";
    contract.SecType = "FUT";
    contract.Exchange = "DTB";
    contract.Currency = "EUR";
    contract.LastTradeDateOrContractMonth = "201609";
    contract.Multiplier = "5";
```

```
Contract contract = new Contract();
contract.symbol("DAX");
contract.secType("FUT");
contract.currency("EUR");
contract.exchange("DTB");
contract.lastTradeDateOrContractMonth("201609");
contract.multiplier("5");
```

```
            Dim contract As Contract = New Contract()
            contract.Symbol = "DAX"
            contract.SecType = "FUT"
            contract.Exchange = "DTB"
            contract.Currency = "EUR"
            contract.LastTradeDateOrContractMonth = "201609"
            contract.Multiplier = "5"
```

```
    Contract contract;
    contract.symbol = "DAX";
    contract.secType = "FUT";
    contract.exchange = "DTB";
    contract.currency = "EUR";
    contract.lastTradeDateOrContractMonth = "201609";
    contract.multiplier = "5";
```

```
        contract = Contract()
        contract.symbol = "DAX"
        contract.secType = "FUT"
        contract.exchange = "DTB"
        contract.currency = "EUR"
        contract.lastTradeDateOrContractMonth = "201609"
        contract.multiplier = "5"
```

A project is under development to allow for **continuous futures** from the API to receive historical data, real time bars, or contract details (requires TWS v966+). Continuous futures cannot be used with real time data or to place orders. To request historical data with continuous futures it is recommended to use TWS Build 971 or higher.

```
            Contract contract = new Contract();
            contract.Symbol = "ES";
            contract.SecType = "CONTFUT";
            contract.Exchange = "GLOBEX";
```

```
        Contract contract = new Contract();
        contract.symbol("ES");
        contract.secType("CONTFUT");
        contract.exchange("GLOBEX");
```

```
            Dim contract As Contract = New Contract()
            contract.Symbol = "ES"
            contract.SecType = "CONTFUT"
            contract.Exchange = "GLOBEX"
```

```
    Contract contract;
    contract.symbol = "ES";
    contract.secType = "CONTFUT";
    contract.exchange = "GLOBEX";
```

```
        contract = Contract()
        contract.symbol = "ES"
        contract.secType = "CONTFUT"
        contract.exchange = "GLOBEX"
```

The security type "FUT+CONTFUT" can be used to request contract details about the futures and continuous futures on an underlying. This security type cannot be used with other functionality.

```
            Contract contract = new Contract();
            contract.Symbol = "ES";
            contract.SecType = "FUT+CONTFUT";
            contract.Exchange = "GLOBEX";
```

```
        Contract contract = new Contract();
        contract.symbol("ES");
        contract.secType("FUT+CONTFUT");
        contract.exchange("GLOBEX");
```

```
            Dim contract As Contract = New Contract()
            contract.Symbol = "ES"
            contract.SecType = "FUT+CONTFUT"
            contract.Exchange = "GLOBEX"
```

```
    Contract contract;
    contract.symbol = "ES";
    contract.secType = "FUT+CONTFUT";
    contract.exchange = "GLOBEX";
```

```
        contract = Contract()
        contract.symbol = "ES"
        contract.secType = "FUT+CONTFUT"
        contract.exchange = "GLOBEX"
```

9.4.6 Options

Options, like futures, also require an expiration date plus a **strike** and a **multiplier**:

- ```
            Contract contract = new Contract();
            contract.Symbol = "GOOG";
            contract.SecType = "OPT";
            contract.Exchange = "BOX";
            contract.Currency = "USD";
            contract.LastTradeDateOrContractMonth = "20170120";
            contract.Strike = 615;
            contract.Right = "C";
            contract.Multiplier = "100";
  ```

- ```
        Contract contract = new Contract();
        contract.symbol("GOOG");
        contract.secType("OPT");
        contract.currency("USD");
        contract.exchange("BOX");
        contract.lastTradeDateOrContractMonth("20170120");
        contract.right("C");
        contract.strike(615);
        contract.multiplier("100");
  ```

- ```
            Dim contract As Contract = New Contract
            contract.Symbol = "GOOG"
            contract.SecType = "OPT"
            contract.Exchange = "BOX"
            contract.Currency = "USD"
            contract.LastTradeDateOrContractMonth = "20170120"
            contract.Strike = 615
            contract.Right = "C"
            contract.Multiplier = "100"
  ```

- ```
    Contract contract;
    contract.symbol = "GOOG";
    contract.secType = "OPT";
    contract.exchange = "BOX";
    contract.currency = "USD";
    contract.lastTradeDateOrContractMonth = "20170120";
    contract.strike = 615;
    contract.right = "C";
    contract.multiplier = "100";
  ```

- ```
        contract = Contract()
        contract.symbol = "GOOG"
        contract.secType = "OPT"
        contract.exchange = "BOX"
        contract.currency = "USD"
        contract.lastTradeDateOrContractMonth = "20170120"
        contract.strike = 615
        contract.right = "C"
        contract.multiplier = "100"
  ```

It is not unusual to find many option contracts with an almost identical description (i.e. underlying symbol, strike, last trading date, multiplier, etc.). Adding more details such as the **trading class** will help:

- ```
            Contract contract = new Contract();
            contract.Symbol = "SANT";
            contract.SecType = "OPT";
            contract.Exchange = "MEFFRV";
            contract.Currency = "EUR";
            contract.LastTradeDateOrContractMonth = "20190621";
            contract.Strike = 7.5;
            contract.Right = "C";
            contract.Multiplier = "100";
            contract.TradingClass = "SANEU";
  ```

- ```
        Contract contract = new Contract();
        contract.symbol("SANT");
        contract.secType("OPT");
        contract.currency("EUR");
        contract.exchange("MEFFRV");
        contract.lastTradeDateOrContractMonth("20190621");
        contract.right("C");
        contract.strike(7.5);
        contract.multiplier("100");
        contract.tradingClass("SANEU");
  ```
  ```

- ```
  Dim contract As Contract = New Contract
  contract.Symbol = "SANT"
  contract.SecType = "OPT"
  contract.Exchange = "MEFFRV"
  contract.Currency = "EUR"
  contract.LastTradeDateOrContractMonth = "20190621"
  contract.Strike = 7.5
  contract.Right = "C"
  contract.Multiplier = "100"
  contract.TradingClass = "SANEU"
  ```

- ```
  Contract contract;
  contract.symbol = "SANT";
  contract.secType = "OPT";
  contract.exchange = "MEFFRV";
  contract.currency = "EUR";
  contract.lastTradeDateOrContractMonth = "20190621";
  contract.strike = 7.5;
  contract.right = "C";
  contract.multiplier = "100";
  contract.tradingClass = "SANEU";
  ```

- ```
  contract = Contract()
  contract.symbol = "SANT"
  contract.secType = "OPT"
  contract.exchange = "MEFFRV"
  contract.currency = "EUR"
  contract.lastTradeDateOrContractMonth = "20190621"
  contract.strike = 7.5
  contract.right = "C"
  contract.multiplier = "100"
  contract.tradingClass = "SANEU"
  ```

The OCC options symbol can be used to define an option contract in the API through the option's 'local symbol' field.

- ```
  Contract contract = new Contract();
  //Watch out for the spaces within the local symbol!
  contract.LocalSymbol = "C DBK  DEC 20  1600";
  contract.SecType = "OPT";
  contract.Exchange = "DTB";
  contract.Currency = "EUR";
  ```

- ```
  Contract contract = new Contract();
  //Watch out for the spaces within the local symbol!
  contract.localSymbol("C DBK  DEC 20  1600");
  contract.secType("OPT");
  contract.exchange("DTB");
  contract.currency("EUR");
  ```

- ```
  Dim contract As Contract = New Contract()
  'Watch out for the spaces within the local symbol!
  contract.LocalSymbol = "C DBK  DEC 20  1600"
  contract.SecType = "OPT"
  contract.Exchange = "DTB"
  contract.Currency = "EUR"
  ```

- ```
  Contract contract;
  //Watch out for the spaces within the local symbol!
  contract.localSymbol = "C DBK  DEC 20  1600";
  contract.secType = "OPT";
  contract.exchange = "DTB";
  contract.currency = "EUR";
  ```

- ```
  contract = Contract()
  #Watch out for the spaces within the local symbol!
  contract.localSymbol = "C DBK  DEC 20  1600"
  contract.secType = "OPT"
  contract.exchange = "DTB"
  contract.currency = "EUR"
  ```

9.4.7 Futures Options

Futures options follow the same rules as conventional option contracts. For some futures options (e.g GE) it will be necessary to define a trading class, or use the local symbol or conId.

```
            Contract contract = new Contract();
            contract.Symbol = "ES";
            contract.SecType = "FOP";
            contract.Exchange = "GLOBEX";
            contract.Currency = "USD";
            contract.LastTradeDateOrContractMonth = "20180316";
            contract.Strike = 2800;
            contract.Right = "C";
            contract.Multiplier = "50";
```

```
        Contract contract = new Contract();
        contract.symbol("ES");
        contract.secType("FOP");
        contract.currency("USD");
        contract.exchange("GLOBEX");
        contract.lastTradeDateOrContractMonth("20180316");
        contract.right("C");
        contract.strike(2800);
        contract.multiplier("50");
```

```
            Dim contract As Contract = New Contract
            contract.Symbol = "ES"
            contract.SecType = "FOP"
            contract.Exchange = "GLOBEX"
            contract.Currency = "USD"
            contract.LastTradeDateOrContractMonth = "20180316"
            contract.Strike = 2800
            contract.Right = "C"
            contract.Multiplier = "50"
```

```
    Contract contract;
    contract.symbol = "ES";
    contract.secType = "FOP";
    contract.exchange = "GLOBEX";
    contract.currency = "USD";
    contract.lastTradeDateOrContractMonth = "20180316";
    contract.strike = 2800;
    contract.right = "C";
    contract.multiplier = "50";
```

```
        contract = Contract()
        contract.symbol = "SPX"
        contract.secType = "FOP"
        contract.exchange = "GLOBEX"
        contract.currency = "USD"
        contract.lastTradeDateOrContractMonth = "20180315"
        contract.strike = 1025
        contract.right = "C"
        contract.multiplier = "250"
```

9.4.8 Bonds

Bonds can be specified by defining the symbol as the CUSIP or ISIN.

```
            Contract contract = new Contract();
            // enter CUSIP as symbol
            contract.Symbol = "912828C57";
            contract.SecType = "BOND";
            contract.Exchange = "SMART";
            contract.Currency = "USD";
```

```
        Contract contract = new Contract();
        // enter CUSIP as symbol
        contract.symbol("912828C57");
        contract.secType("BOND");
        contract.exchange("SMART");
        contract.currency("USD");
```

```
            Dim Contract As Contract = New Contract
            ' enter CUSIP as symbol
            contract.symbol= "912828C57"
            contract.secType = "BOND"
            contract.exchange = "SMART"
            contract.currency = "USD"
```

```
    Contract contract;
    // enter CUSIP as symbol
    contract.symbol= "912828C57";
    contract.secType = "BOND";
    contract.exchange = "SMART";
    contract.currency = "USD";
```

- ```
  contract = Contract()
  # enter CUSIP as symbol
  contract.symbol= "912828C57"
  contract.secType = "BOND"
  contract.exchange = "SMART"
  contract.currency = "USD"
  ```

Bonds can also be defined with the conId and exchange as with any security type.

- ```
  Contract contract = new Contract();
  contract.ConId = 285191782;
  contract.Exchange = "SMART";
  ```

- ```
  Contract contract = new Contract();
  contract.conid(285191782);
  contract.exchange("SMART");
  ```

- ```
  Dim Contract As Contract = New Contract
  Contract.ConId = 285191782
  Contract.exchange = "SMART"
  ```

- ```
  Contract contract;
  contract.conId = 285191782;
  contract.exchange = "SMART";
  ```

- ```
  contract = Contract()
  contract.conId = 15960357
  contract.exchange = "SMART"
  ```

9.4.9 Mutual Funds

Trading Mutual Funds is **not** currently fully-supported from the API. Note: Mutual Funds orders cannot be placed in paper accounts from any trading system.

- ```
  Contract contract = new Contract();
  contract.Symbol = "VINIX";
  contract.SecType = "FUND";
  contract.Exchange = "FUNDSERV";
  contract.Currency = "USD";
  ```

- ```
  Contract contract = new Contract();
  contract.symbol("VINIX");
  contract.secType("FUND");
  contract.exchange("FUNDSERV");
  contract.currency("USD");
  ```

- ```
  Dim Contract As Contract = New Contract
  Contract.symbol = "VINIX"
  Contract.secType = "FUND"
  Contract.exchange = "FUNDSERV"
  Contract.currency = "USD"
  ```

- ```
  Contract contract;
  contract.symbol = "VINIX";
  contract.secType = "FUND";
  contract.exchange = "FUNDSERV";
  contract.currency = "USD";
  ```

- ```
  contract = Contract()
  contract.symbol = "VINIX"
  contract.secType = "FUND"
  contract.exchange = "FUNDSERV"
  contract.currency = "USD"
  ```
  ```

9.4.10 Commodities

```
            Contract contract = new Contract();
            contract.Symbol = "XAUUSD";
            contract.SecType = "CMDTY";
            contract.Exchange = "SMART";
            contract.Currency = "USD";
```

```
        Contract contract = new Contract();
        contract.symbol("XAUUSD");
        contract.secType("CMDTY");
        contract.exchange("SMART");
        contract.currency("USD");
```

```
            Dim Contract As Contract = New Contract
            Contract.symbol = "XAUUSD"
            Contract.secType = "CMDTY"
            Contract.exchange = "SMART"
            Contract.currency = "USD"
```

```
    Contract contract;
    contract.symbol = "XAUUSD";
    contract.secType = "CMDTY";
    contract.exchange = "SMART";
    contract.currency = "USD";
```

```
            contract = Contract()
            contract.symbol = "XAUUSD"
            contract.secType = "CMDTY"
            contract.exchange = "SMART"
            contract.currency = "USD"
```

9.4.11 Dutch Warrants and Structured Products

To unambiguously define a Dutch Warrant or Structured Product (IOPTs) the conId or localSymbol field must be used.

- It is important to note that if reqContractDetails is used with an incompletely-defined IOPT contract definition, that thousands of results can be returned and the API connection broken.
- IOPT contract definitions will often change and it will be necessary to restart TWS or IB Gateway to download the new contract definition.

```
            Contract contract = new Contract();
            contract.LocalSymbol = "B881G";
            contract.SecType = "IOPT";
            contract.Exchange = "SBF";
            contract.Currency = "EUR";
```

```
        Contract contract = new Contract();
        contract.localSymbol("B881G");
        contract.secType("IOPT");
        contract.exchange("SBF");
        contract.currency("EUR");
```

```
            Dim contract As Contract = New Contract()
            contract.LocalSymbol = "B881G"
            contract.SecType = "IOPT"
            contract.Exchange = "SBF"
            contract.Currency = "EUR"
```

```
    Contract contract;
    contract.localSymbol = "B881G";
    contract.secType = "IOPT";
    contract.exchange = "SBF";
    contract.currency = "EUR";
```

```
        contract = Contract()
        contract.localSymbol = "B881G"
        contract.secType = "IOPT"
        contract.exchange = "SBF"
        contract.currency = "EUR"
```

9.5 Spreads

Spread contracts, also known as combos or combinations, combine two or more instruments. To define a combination contract it is required to know the conId of the **IBApi.Contract** in question. The conId of an instrument can easily be obtained via the **IBApi.EClientSocket.reqContractDetails** request.

The spread contract's symbol can be either the symbol of one of the contract legs or, for two-legged combinations the symbols of both legs separated by a comma as shown in the examples below.

9.5.1 Stock Spread

Beginning with TWS v971, Stock/Stock combos will have ticker symbols in alphabetical order when they are both used in the symbol field, e.g. "AMD,IBKR"

- ```
            Contract contract = new Contract();
            contract.Symbol = "IBKR,MCD";
            contract.SecType = "BAG";
            contract.Currency = "USD";
            contract.Exchange = "SMART";

            ComboLeg leg1 = new ComboLeg();
            leg1.ConId = 43645865;//IBKR STK
            leg1.Ratio = 1;
            leg1.Action = "BUY";
            leg1.Exchange = "SMART";

            ComboLeg leg2 = new ComboLeg();
            leg2.ConId = 9408;//MCD STK
            leg2.Ratio = 1;
            leg2.Action = "SELL";
            leg2.Exchange = "SMART";

            contract.ComboLegs = new List<ComboLeg>();
            contract.ComboLegs.Add(leg1);
            contract.ComboLegs.Add(leg2);
  ```
- ```
        Contract contract = new Contract();
        contract.symbol("MCD");
        contract.secType("BAG");
        contract.currency("USD");
        contract.exchange("SMART");

        ComboLeg leg1 = new ComboLeg();
        ComboLeg leg2 = new ComboLeg();

        List<ComboLeg> addAllLegs = new ArrayList<>();

        leg1.conid(43645865);//IBKR STK
        leg1.ratio(1);
        leg1.action("BUY");
        leg1.exchange("SMART");

        leg2.conid(9408);//MCD STK
        leg2.ratio(1);
        leg2.action("SELL");
        leg2.exchange("SMART");

        addAllLegs.add(leg1);
        addAllLegs.add(leg2);

        contract.comboLegs(addAllLegs);
  ```
- ```
            Dim contract As Contract = New Contract
            contract.Symbol = "MCD"
            contract.SecType = "BAG"
            contract.Currency = "USD"
            contract.Exchange = "SMART"

            Dim leg1 As ComboLeg = New ComboLeg
            leg1.ConId = 43645865
            leg1.Ratio = 1
            leg1.Action = "BUY"
            leg1.Exchange = "SMART"

            Dim leg2 As ComboLeg = New ComboLeg
            leg2.ConId = 9408
  ```
  ```

```
            leg2.Ratio = 1
            leg2.Action = "SELL"
            leg2.Exchange = "SMART"

            contract.ComboLegs = New List(Of ComboLeg)
            contract.ComboLegs.Add(leg1)
            contract.ComboLegs.Add(leg2)
```

- ```
  Contract contract;
  contract.symbol = "MCD";
  contract.secType = "BAG";
  contract.currency = "USD";
  contract.exchange = "SMART";

  ComboLegSPtr leg1(new ComboLeg);
  leg1->conId = 43645865;
  leg1->action = "BUY";
  leg1->ratio = 1;
  leg1->exchange = "SMART";

  ComboLegSPtr leg2(new ComboLeg);
  leg2->conId = 9408;
  leg2->action = "SELL";
  leg2->ratio = 1;
  leg2->exchange = "SMART";

  contract.comboLegs.reset(new Contract::ComboLegList());
  contract.comboLegs->push_back(leg1);
  contract.comboLegs->push_back(leg2);
  ```

- ```
      contract = Contract()
      contract.symbol = "IBKR,MCD"
      contract.secType = "BAG"
      contract.currency = "USD"
      contract.exchange = "SMART"

      leg1 = ComboLeg()
      leg1.conId = 43645865#IBKR STK
      leg1.ratio = 1
      leg1.action = "BUY"
      leg1.exchange = "SMART"

      leg2 = ComboLeg()
      leg2.conId = 9408#MCD STK
      leg2.ratio = 1
      leg2.action = "SELL"
      leg2.exchange = "SMART"

      contract.comboLegs = []
      contract.comboLegs.append(leg1)
      contract.comboLegs.append(leg2)
  ```

Note: EFPs are simply defined as a bag contract of stock and corresponding SSF with a ratio of 100:1.

9.5.2 Options Spread

- ```
            Contract contract = new Contract();
            contract.Symbol = "DBK";
            contract.SecType = "BAG";
            contract.Currency = "EUR";
            contract.Exchange = "DTB";

            ComboLeg leg1 = new ComboLeg();
            leg1.ConId = 197397509;//DBK JUN 15 '18 C
            leg1.Ratio = 1;
            leg1.Action = "BUY";
            leg1.Exchange = "DTB";

            ComboLeg leg2 = new ComboLeg();
            leg2.ConId = 197397584;//DBK JUN 15 '18 P
            leg2.Ratio = 1;
            leg2.Action = "SELL";
            leg2.Exchange = "DTB";

            contract.ComboLegs = new List<ComboLeg>();
            contract.ComboLegs.Add(leg1);
            contract.ComboLegs.Add(leg2);
  ```
  ```

-
```
Contract contract = new Contract();
contract.symbol("DBK");
contract.secType("BAG");
contract.currency("EUR");
contract.exchange("DTB");

ComboLeg leg1 = new ComboLeg();
ComboLeg leg2 = new ComboLeg();

List<ComboLeg> addAllLegs = new ArrayList<>();

leg1.conid(197397509);//DBK JUN 15 '18 C
leg1.ratio(1);
leg1.action("BUY");
leg1.exchange("DTB");

leg2.conid(197397584);//DBK JUN 15 '18 P
leg2.ratio(1);
leg2.action("SELL");
leg2.exchange("DTB");

addAllLegs.add(leg1);
addAllLegs.add(leg2);

contract.comboLegs(addAllLegs);
```

-
```
    Dim contract As Contract = New Contract
    contract.Symbol = "DBK"
    contract.SecType = "BAG"
    contract.Currency = "EUR"
    contract.Exchange = "DTB"

    Dim leg1 As ComboLeg = New ComboLeg
    leg1.ConId = 197397509 'DBK JUN 15 '18 C
    leg1.Ratio = 1
    leg1.Action = "BUY"
    leg1.Exchange = "DTB"

    Dim leg2 As ComboLeg = New ComboLeg
    leg2.ConId = 197397584 'DBK JUN 15 '18 P
    leg2.Ratio = 1
    leg2.Action = "SELL"
    leg2.Exchange = "DTB"

    contract.ComboLegs = New List(Of ComboLeg)
    contract.ComboLegs.Add(leg1)
    contract.ComboLegs.Add(leg2)
```

-
```
Contract contract;
contract.symbol = "DBK";
contract.secType = "BAG";
contract.currency = "EUR";
contract.exchange = "DTB";

ComboLegSPtr leg1(new ComboLeg);
leg1->conId = 197397509;
leg1->action = "BUY";
leg1->ratio = 1;
leg1->exchange = "DTB";

ComboLegSPtr leg2(new ComboLeg);
leg2->conId = 197397584;
leg2->action = "SELL";
leg2->ratio = 1;
leg2->exchange = "DTB";

contract.comboLegs.reset(new Contract::ComboLegList());
contract.comboLegs->push_back(leg1);
contract.comboLegs->push_back(leg2);
```

-
```
    contract = Contract()
    contract.symbol = "DBK"
    contract.secType = "BAG"
    contract.currency = "EUR"
    contract.exchange = "DTB"

    leg1 = ComboLeg()
    leg1.conId = 197397509 #DBK JUN 15 2018 C
    leg1.ratio = 1
    leg1.action = "BUY"
    leg1.exchange = "DTB"

    leg2 = ComboLeg()
    leg2.conId = 197397584  #DBK JUN 15 2018 P
    leg2.ratio = 1
    leg2.action = "SELL"
    leg2.exchange = "DTB"
```

```
contract.comboLegs = []
contract.comboLegs.append(leg1)
contract.comboLegs.append(leg2)
```

9.5.3 Guaranteed Futures Spread

- ```
  Contract contract = new Contract();
  contract.Symbol = "VIX";
  contract.SecType = "BAG";
  contract.Currency = "USD";
  contract.Exchange = "CFE";

  ComboLeg leg1 = new ComboLeg();
  leg1.ConId = 195538625;//VIX FUT 20160217
  leg1.Ratio = 1;
  leg1.Action = "BUY";
  leg1.Exchange = "CFE";

  ComboLeg leg2 = new ComboLeg();
  leg2.ConId = 197436571;//VIX FUT 20160316
  leg2.Ratio = 1;
  leg2.Action = "SELL";
  leg2.Exchange = "CFE";

  contract.ComboLegs = new List<ComboLeg>();
  contract.ComboLegs.Add(leg1);
  contract.ComboLegs.Add(leg2);
  ```

- ```
  Contract contract = new Contract();
  contract.symbol("VIX");
  contract.secType("BAG");
  contract.currency("USD");
  contract.exchange("CFE");

  ComboLeg leg1 = new ComboLeg();
  ComboLeg leg2 = new ComboLeg();

  List<ComboLeg> addAllLegs = new ArrayList<>();

  leg1.conid(195538625);//VIX FUT 20160217
  leg1.ratio(1);
  leg1.action("BUY");
  leg1.exchange("CFE");

  leg2.conid(197436571);//VIX FUT 20160316
  leg2.ratio(1);
  leg2.action("SELL");
  leg2.exchange("CFE");

  addAllLegs.add(leg1);
  addAllLegs.add(leg2);

  contract.comboLegs(addAllLegs);
  ```

- ```
  Dim contract As Contract = New Contract
  contract.Symbol = "VIX"
  contract.SecType = "BAG"
  contract.Currency = "USD"
  contract.Exchange = "CFE"

  Dim leg1 As ComboLeg = New ComboLeg
  leg1.ConId = 195538625
  leg1.Ratio = 1
  leg1.Action = "BUY"
  leg1.Exchange = "CFE"

  Dim leg2 As ComboLeg = New ComboLeg
  leg2.ConId = 197436571
  leg2.Ratio = 1
  leg2.Action = "SELL"
  leg2.Exchange = "CFE"

  contract.ComboLegs = New List(Of ComboLeg)
  contract.ComboLegs.Add(leg1)
  contract.ComboLegs.Add(leg2)
  ```

- ```
  Contract contract;
  contract.symbol = "VIX";
  contract.secType = "BAG";
  contract.currency = "USD";
  contract.exchange = "CFE";
  ```

```
ComboLegSPtr leg1(new ComboLeg);
leg1->conId = 195538625;
leg1->action = "BUY";
leg1->ratio = 1;
leg1->exchange = "CFE";

ComboLegSPtr leg2(new ComboLeg);
leg2->conId = 197436571;
leg2->action = "SELL";
leg2->ratio = 1;
leg2->exchange = "CFE";

contract.comboLegs.reset(new Contract::ComboLegList());
contract.comboLegs->push_back(leg1);
contract.comboLegs->push_back(leg2);
```

- ```
  contract = Contract()
  contract.symbol = "VIX"
  contract.secType = "BAG"
  contract.currency = "USD"
  contract.exchange = "CFE"

  leg1 = ComboLeg()
  leg1.conId = 256038899 # VIX FUT 201708
  leg1.ratio = 1
  leg1.action = "BUY"
  leg1.exchange = "CFE"

  leg2 = ComboLeg()
  leg2.conId = 260564703 # VIX FUT 201709
  leg2.ratio = 1
  leg2.action = "SELL"
  leg2.exchange = "CFE"

  contract.comboLegs = []
  contract.comboLegs.append(leg1)
  contract.comboLegs.append(leg2)
  ```

9.5.4 Smart-Routed Futures Spread

Futures spreads can also be defined as Smart-routed (non-guaranteed) combos. When placing an order for a non-guaranteed combo from the API, the non-guaranteed flag must be set to 1. Historical data for smart-routed futures spreads is generally available from the API with the requisite market data subscriptions.

- ```
  Contract contract = new Contract();
  contract.Symbol = "WTI"; // WTI,COIL spread. Symbol can be defined as first leg symbol ("WTI")
or currency ("USD").
  contract.SecType = "BAG";
  contract.Currency = "USD";
  contract.Exchange = "SMART";

  ComboLeg leg1 = new ComboLeg();
  leg1.ConId = 55928698;//WTI future June 2017
  leg1.Ratio = 1;
  leg1.Action = "BUY";
  leg1.Exchange = "IPE";

  ComboLeg leg2 = new ComboLeg();
  leg2.ConId = 55850663;//COIL future June 2017
  leg2.Ratio = 1;
  leg2.Action = "SELL";
  leg2.Exchange = "IPE";

  contract.ComboLegs = new List<ComboLeg>();
  contract.ComboLegs.Add(leg1);
  contract.ComboLegs.Add(leg2);
  ```

- ```
  Contract contract = new Contract();
  contract.symbol("WTI");  // WTI,COIL spread. Symbol can be defined as first leg symbol ("WTI") or
currency ("USD").
  contract.secType("BAG");
  contract.currency("USD");
  contract.exchange("SMART"); // smart-routed rather than direct routed

  ComboLeg leg1 = new ComboLeg();
  ComboLeg leg2 = new ComboLeg();

  List<ComboLeg> addAllLegs = new ArrayList<>();

  leg1.conid(55928698);// WTI future June 2017
  ```
  ```

```
leg1.ratio(1);
leg1.action("BUY");
leg1.exchange("IPE");

leg2.conid(55850663);// COIL future June 2017
leg2.ratio(1);
leg2.action("SELL");
leg2.exchange("IPE");

addAllLegs.add(leg1);
addAllLegs.add(leg2);

contract.comboLegs(addAllLegs);
```

- ```
    Dim contract As Contract = New Contract
    contract.Symbol = "WTI" ' WTI,COIL spread. Symbol can be defined as first leg symbol ("WTI") or currency ("USD")
    contract.SecType = "BAG"
    contract.Currency = "USD"
    contract.Exchange = "SMART"

    Dim leg1 As ComboLeg = New ComboLeg
    leg1.ConId = 55928698 ' WTI future June 2017
    leg1.Ratio = 1
    leg1.Action = "BUY"
    leg1.Exchange = "IPE"

    Dim leg2 As ComboLeg = New ComboLeg
    leg2.ConId = 55850663 ' COIL future June 2017
    leg2.Ratio = 1
    leg2.Action = "SELL"
    leg2.Exchange = "IPE"

    contract.ComboLegs = New List(Of ComboLeg)
    contract.ComboLegs.Add(leg1)
    contract.ComboLegs.Add(leg2)
  ```

- ```
  Contract contract;
  contract.symbol = "WTI"; // WTI,COIL spread. Symbol can be defined as first leg symbol ("WTI") or currency ("USD").
  contract.secType = "BAG";
  contract.currency = "USD";
  contract.exchange = "SMART";

  ComboLegSPtr leg1(new ComboLeg);
  leg1->conId = 55928698; // WTI future June 2017
  leg1->action = "BUY";
  leg1->ratio = 1;
  leg1->exchange = "IPE";

  ComboLegSPtr leg2(new ComboLeg);
  leg2->conId = 55850663; // COIL future June 2017
  leg2->action = "SELL";
  leg2->ratio = 1;
  leg2->exchange = "IPE";

  contract.comboLegs.reset(new Contract::ComboLegList());
  contract.comboLegs->push_back(leg1);
  contract.comboLegs->push_back(leg2);
  ```

- ```
    contract = Contract()
    contract.symbol = "WTI" # WTI,COIL spread. Symbol can be defined as first leg symbol ("WTI") or currency ("USD")
    contract.secType = "BAG"
    contract.currency = "USD"
    contract.exchange = "SMART"

    leg1 = ComboLeg()
    leg1.conId = 55928698 # WTI future June 2017
    leg1.ratio = 1
    leg1.action = "BUY"
    leg1.exchange = "IPE"

    leg2 = ComboLeg()
    leg2.conId = 55850663 # COIL future June 2017
    leg2.ratio = 1
    leg2.action = "SELL"
    leg2.exchange = "IPE"

    contract.comboLegs = []
    contract.comboLegs.append(leg1)
    contract.comboLegs.append(leg2)
  ```
  ```

9.5.5 Inter-Commodity Futures

For Inter-Commodity futures, the 'Local Symbol' field in TWS is used for the 'Symbol' field in the API contract definition, e.g. "CL.BZ". They are always guaranteed combos, which is the default in the API.

- ```
  Contract contract = new Contract();
  contract.Symbol = "CL.BZ";
  contract.SecType = "BAG";
  contract.Currency = "USD";
  contract.Exchange = "NYMEX";

  ComboLeg leg1 = new ComboLeg();
  leg1.ConId = 47207310; //CL Dec'16 @NYMEX
  leg1.Ratio = 1;
  leg1.Action = "BUY";
  leg1.Exchange = "NYMEX";

  ComboLeg leg2 = new ComboLeg();
  leg2.ConId = 47195961; //BZ Dec'16 @NYMEX
  leg2.Ratio = 1;
  leg2.Action = "SELL";
  leg2.Exchange = "NYMEX";

  contract.ComboLegs = new List<ComboLeg>();
  contract.ComboLegs.Add(leg1);
  contract.ComboLegs.Add(leg2);
  ```
- ```
  Contract contract = new Contract();
  contract.symbol("CL.BZ");
  contract.secType("BAG");
  contract.currency("USD");
  contract.exchange("NYMEX");

  ComboLeg leg1 = new ComboLeg();
  ComboLeg leg2 = new ComboLeg();

  List<ComboLeg> addAllLegs = new ArrayList<>();

  leg1.conid(47207310); //CL Dec'16 @NYMEX
  leg1.ratio(1);
  leg1.action("BUY");
  leg1.exchange("NYMEX");

  leg2.conid(47195961); //BZ Dec'16 @NYMEX
  leg2.ratio(1);
  leg2.action("SELL");
  leg2.exchange("NYMEX");

  addAllLegs.add(leg1);
  addAllLegs.add(leg2);

  contract.comboLegs(addAllLegs);
  ```
- ```
  Dim contract As Contract = New Contract
  contract.Symbol = "CL.BZ"
  contract.SecType = "BAG"
  contract.Currency = "USD"
  contract.Exchange = "NYMEX"

  Dim leg1 As ComboLeg = New ComboLeg
  leg1.ConId = 47207310 ' CL Dec'16 @NYMEX
  leg1.Ratio = 1
  leg1.Action = "BUY"
  leg1.Exchange = "NYMEX"

  Dim leg2 As ComboLeg = New ComboLeg
  leg2.ConId = 47195961 ' BZ Dec'16 @NYMEX
  leg2.Ratio = 1
  leg2.Action = "SELL"
  leg2.Exchange = "NYMEX"

  contract.ComboLegs = New List(Of ComboLeg)
  contract.ComboLegs.Add(leg1)
  contract.ComboLegs.Add(leg2)
  ```
- ```
  Contract contract;
  contract.symbol = "CL.BZ";
  contract.secType = "BAG";
  contract.currency = "USD";
  contract.exchange = "NYMEX";

  ComboLegSPtr leg1(new ComboLeg);
  ```

```
leg1->conId = 47207310; //CL Dec'16 @NYMEX
leg1->action = "BUY";
leg1->ratio = 1;
leg1->exchange = "NYMEX";

ComboLegSPtr leg2(new ComboLeg);
leg2->conId = 47195961; //BZ Dec'16 @NYMEX
leg2->action = "SELL";
leg2->ratio = 1;
leg2->exchange = "NYMEX";

contract.comboLegs.reset(new Contract::ComboLegList());
contract.comboLegs->push_back(leg1);
contract.comboLegs->push_back(leg2);
```

- ```
  contract = Contract()
  contract.symbol = "CL.BZ" #symbol is 'local symbol' of intercommodity spread.
  contract.secType = "BAG"
  contract.currency = "USD"
  contract.exchange = "NYMEX"

  leg1 = ComboLeg()
  leg1.conId = 47207310 #CL Dec'16 @NYMEX
  leg1.ratio = 1
  leg1.action = "BUY"
  leg1.exchange = "NYMEX"

  leg2 = ComboLeg()
  leg2.conId = 47195961 #BZ Dec'16 @NYMEX
  leg2.ratio = 1
  leg2.action = "SELL"
  leg2.exchange = "NYMEX"

  contract.comboLegs = []
  contract.comboLegs.append(leg1)
  contract.comboLegs.append(leg2)
  ```

Please be mindful of the fact that inter-commodity spreads are offered by the exchange directly, and so they are direct-routed though the legs have different underlyings. Only real time, and not historical, data is offered for inter-commodity spread contracts through the API.

It is also possible in many cases to create a spread of the same future contracts in a inter-commodity spread which is smart-routed and non-guaranteed. Historical data for this spread would generally be available from the API. Also, historical data for expired spread contracts is not available in TWS or the API.
  ```

Chapter 9

Orders

http://interactivebrokers.github.io/tws-api/orders.html

10.1 Overview

Through the TWS API it is possible to define most orders already available within the TWS.

- **Available Orders**
- **Order Management**
- **Minimum Price Increment**
- **Checking Margin Changes**

10.2 Available Orders

- **Basic Orders**
- **Advanced Orders and Algos**

10.2.1 Basic Orders

10.2.1.1 Auction

An `Auction` order is entered into the electronic trading system during the pre-market opening period for execution at the Calculated Opening Price (COP). If your order is not filled on the open, the order is re-submitted as a limit order with the limit price set to the COP or the best bid/ask after the market opens.

- Products: FUT, STK
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.Tif = "AUC";
            order.OrderType = "MTL";
            order.TotalQuantity = quantity;
            order.LmtPrice = price;
```

```
        Order order = new Order();
        order.action(action);
        order.tif("AUC");
        order.orderType("MTL");
        order.totalQuantity(quantity);
        order.lmtPrice(price);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.Tif = "AUC"
            order.OrderType = "MTL"
            order.TotalQuantity = quantity
            order.LmtPrice = price
```

```
    Order order;
    order.action = action;
    order.tif = "AUC";
    order.orderType = "MTL";
    order.totalQuantity = quantity;
    order.lmtPrice = price;
```

```
        order = Order()
        order.action = action
        order.tif = "AUC"
        order.orderType = "MTL"
        order.totalQuantity = quantity
        order.lmtPrice = price
```

10.2.1.2 Discretionary

An `Discretionary` order is a limit order submitted with a hidden, specified 'discretionary' amount off the limit price which may be used to increase the price range over which the limit order is eligible to execute. The market sees only the limit price.

- Products: STK
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "LMT";
            order.TotalQuantity = quantity;
            order.LmtPrice = price;
            order.DiscretionaryAmt = discretionaryAmount;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("LMT");
        order.totalQuantity(quantity);
        order.lmtPrice(price);
        order.discretionaryAmt(discretionaryAmt);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LMT"
            order.TotalQuantity = quantity
            order.LmtPrice = price
            order.DiscretionaryAmt = discretionaryAmount
```

- ```
  Order order;
  order.action = action;
  order.orderType = "LMT";
  order.totalQuantity = quantity;
  order.lmtPrice = price;
  order.discretionaryAmt = discretionaryAmount;
  ```

- ```
  order = Order()
  order.action = action
  order.orderType = "LMT"
  order.totalQuantity = quantity
  order.lmtPrice = price
  order.discretionaryAmt = discretionaryAmount
  ```

10.2.1.3 Market

A `Market` order is an order to buy or sell at the market bid or offer price. A market order may increase the likelihood of a fill and the speed of execution, but unlike the Limit order a Market order provides no price protection and may fill at a price far lower/higher than the current displayed bid/ask.

- Products: BOND, CFD, EFP, CASH, FUND, FUT, FOP, OPT, STK, WAR
- `Supported exchanges`

- ```
  Order order = new Order();
  order.Action = action;
  order.OrderType = "MKT";
  order.TotalQuantity = quantity;
  ```

- ```
  Order order = new Order();
  order.action(action);
  order.orderType("MKT");
  order.totalQuantity(quantity);
  ```

- ```
  Dim order As Order = New Order
  order.Action = action
  order.OrderType = "MKT"
  order.TotalQuantity = quantity
  ```

- ```
  Order order;
  order.action = action;
  order.orderType = "MKT";
  order.totalQuantity = quantity;
  ```

- ```
  order = Order()
  order.action = action
  order.orderType = "MKT"
  order.totalQuantity = quantity
  ```

10.2.1.4 Market If Touched

A `Market If Touched` (MIT) is an order to buy (or sell) a contract below (or above) the market. Its purpose is to take advantage of sudden or unexpected changes in share or other prices and provides investors with a trigger price to set an order in motion. Investors may be waiting for excessive strength (or weakness) to cease, which might be represented by a specific price point. MIT orders can be used to determine whether or not to enter the market once a specific price level has been achieved. This order is held in the system until the trigger price is touched, and is then submitted as a market order. An MIT order is similar to a stop order, except that an MIT sell order is placed above the current market price, and a stop sell order is placed below

- Products: BOND, CFD, CASH, FUT, FOP, OPT, STK, WAR
- `Supported exchanges`
  ```

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "MIT";
            order.TotalQuantity = quantity;
            order.AuxPrice = price;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("MIT");
        order.totalQuantity(quantity);
        order.auxPrice(price);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "MIT"
            order.TotalQuantity = quantity
            order.AuxPrice = price
```

```
    Order order;
    order.action = action;
    order.orderType = "MIT";
    order.totalQuantity = quantity;
    order.auxPrice = price;
```

```
        order = Order()
        order.action = action
        order.orderType = "MIT"
        order.totalQuantity = quantity
        order.auxPrice = price
```

10.2.1.5 Market On Close

A `Market On Close` (MOC) order is a market order that is submitted to execute as close to the closing price as possible.

- Products: CFD, FUT, STK, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "MOC";
            order.TotalQuantity = quantity;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("MOC");
        order.totalQuantity(quantity);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "MOC"
            order.TotalQuantity = quantity
```

```
    Order order;
    order.action = action;
    order.orderType = "MOC";
    order.totalQuantity = quantity;
```

```
        order = Order()
        order.action = action
        order.orderType = "MOC"
        order.totalQuantity = quantity
```

10.2.1.6 Market On Open

A `Market On Open` (MOO) combines a market order with the OPG time in force to create an order that is automatically submitted at the market's open and fills at the market price.

- Products: CFD, FUT, STK, WAR
- `Supported exchanges`

- ```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "MKT";
            order.TotalQuantity = quantity;
            order.Tif = "OPG";
  ```
- ```
        Order order = new Order();
        order.action(action);
        order.orderType("MKT");
        order.totalQuantity(quantity);
        order.tif("OPG");
  ```
- ```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "MKT"
            order.TotalQuantity = quantity
            order.Tif = "OPG"
  ```
- ```
    Order order;
    order.action = action;
    order.orderType = "MKT";
    order.totalQuantity = quantity;
    order.tif = "OPG";
  ```
- ```
        order = Order()
        order.action = action
        order.orderType = "MKT"
        order.totalQuantity = quantity
        order.tif = "OPG"
  ```

10.2.1.7 Pegged to Market

A `pegged-to-market order` is designed to maintain a purchase price relative to the national best offer (N↩BO) or a sale price relative to the national best bid (NBB). Depending on the width of the quote, this order may be passive or aggressive. The trader creates the order by entering a limit price which defines the worst limit price that they are willing to accept. Next, the trader enters an offset amount which computes the active limit price as follows: Sell order price = Bid price + offset amount Buy order price = Ask price - offset amount

- Products: STK
- `Supported exchanges`

- ```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "PEG MKT";
            order.TotalQuantity = 100;
            order.AuxPrice = marketOffset;//Offset price
  ```
- ```
        Order order = new Order();
        order.action(action);
        order.orderType("PEG MKT");
        order.totalQuantity(100);
        order.auxPrice(marketOffset);//Offset price
  ```
  ```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "PEG MKT"
            order.TotalQuantity = 100
            order.AuxPrice = marketOffset ' Offset price
```

```
    Order order;
    order.action = action;
    order.orderType = "PEG MKT";
    order.totalQuantity = quantity;
    order.auxPrice = marketOffset;
```

```
        order = Order()
        order.action = action
        order.orderType = "PEG MKT"
        order.totalQuantity = quantity
        order.auxPrice = marketOffset#Offset price
```

10.2.1.8 Pegged to Stock

A `Pegged to Stock` order continually adjusts the option order price by the product of a signed user-define delta and the change of the option's underlying stock price. The delta is entered as an absolute and assumed to be positive for calls and negative for puts. A buy or sell call order price is determined by adding the delta times a change in an underlying stock price to a specified starting price for the call. To determine the change in price, the stock reference price is subtracted from the current NBBO midpoint. The Stock Reference Price can be defined by the user, or defaults to the NBBO midpoint at the time of the order if no reference price is entered. You may also enter a high/low stock price range which cancels the order when reached. The delta times the change in stock price will be rounded to the nearest penny in favor of the order.

- Products: OPT
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "PEG STK";
            order.TotalQuantity = quantity;
            order.Delta = delta;
            order.LmtPrice = stockReferencePrice;
            order.StartingPrice = startingPrice;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("PEG STK");
        order.totalQuantity(quantity);
        order.delta(delta);
        order.lmtPrice(stockReferencePrice);
        order.startingPrice(startingPrice);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "PEG STK"
            order.TotalQuantity = quantity
            order.Delta = delta
            order.LmtPrice = stockReferencePrice
            order.StartingPrice = startingPrice
```

```
    Order order;
    order.action = action;
    order.orderType = "PEG STK";
    order.totalQuantity = quantity;
    order.delta = delta;
    order.lmtPrice = stockReferencePrice;
    order.startingPrice = startingPrice;
```

```
        order = Order()
        order.action = action
        order.orderType = "PEG STK"
        order.totalQuantity = quantity
        order.delta = delta
        order.lmtPrice = stockReferencePrice
        order.startingPrice = startingPrice
```

10.2.1.9 Pegged to Primary

`Relative` (a.k.a. Pegged-to-Primary) orders provide a means for traders to seek a more aggressive price than the National Best Bid and Offer (NBBO). By acting as liquidity providers, and placing more aggressive bids and offers than the current best bids and offers, traders increase their odds of filling their order. Quotes are automatically adjusted as the markets move, to remain aggressive. For a buy order, your bid is pegged to the NBB by a more aggressive offset, and if the NBB moves up, your bid will also move up. If the NBB moves down, there will be no adjustment because your bid will become even more aggressive and execute. For sales, your offer is pegged to the NBO by a more aggressive offset, and if the NBO moves down, your offer will also move down. If the NBO moves up, there will be no adjustment because your offer will become more aggressive and execute. In addition to the offset, you can define an absolute cap, which works like a limit price, and will prevent your order from being executed above or below a specified level. Stocks, Options and Futures - not available on paper trading

- Products: CFD, STK, OPT, FUT
- `Supported exchanges`

- ```
  Order order = new Order();
  order.Action = action;
  order.OrderType = "REL";
  order.TotalQuantity = quantity;
  order.LmtPrice = priceCap;
  order.AuxPrice = offsetAmount;
  ```

- ```
  Order order = new Order();
  order.action(action);
  order.orderType("REL");
  order.totalQuantity(quantity);
  order.lmtPrice(priceCap);
  order.auxPrice(offsetAmount);
  ```

- ```
  Dim order As Order = New Order
  order.Action = action
  order.OrderType = "REL"
  order.TotalQuantity = quantity
  order.LmtPrice = priceCap
  order.AuxPrice = offsetAmount
  ```

- ```
  Order order;
  order.action = action;
  order.orderType = "REL";
  order.totalQuantity = quantity;
  order.lmtPrice = priceCap;
  order.auxPrice = offsetAmount;
  ```

- ```
  order = Order()
  order.action = action
  order.orderType = "REL"
  order.totalQuantity = quantity
  order.lmtPrice = priceCap
  order.auxPrice = offsetAmount
  ```

10.2.1.10 Sweep to Fill

`Sweep-to-fill` orders are useful when a trader values speed of execution over price. A sweep-to-fill order identifies the best price and the exact quantity offered/available at that price, and transmits the corresponding portion of your order for immediate execution. Simultaneously it identifies the next best price and quantity offered/available, and submits the matching quantity of your order for immediate execution.

- Products: CFD, STK, WAR (SMART only)

- ```
  Order order = new Order();
  order.Action = action;
  order.OrderType = "LMT";
  order.TotalQuantity = quantity;
  order.LmtPrice = price;
  order.SweepToFill = true;
  ```

```
        Order order = new Order();
        order.action(action);
        order.orderType("LMT");
        order.totalQuantity(quantity);
        order.lmtPrice(price);
        order.sweepToFill(true);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LMT"
            order.TotalQuantity = quantity
            order.LmtPrice = price
            order.SweepToFill = True
```

```
    Order order;
    order.action = action;
    order.orderType = "LMT";
    order.totalQuantity = quantity;
    order.lmtPrice = price;
    order.sweepToFill = true;
```

```
        order = Order()
        order.action = action
        order.orderType = "LMT"
        order.totalQuantity = quantity
        order.lmtPrice = price
        order.sweepToFill = True
```

10.2.1.11 Auction Limit

For option orders routed to the Boston Options Exchange (BOX) you may elect to participate in the BOX's price improvement auction in pennies. All BOX-directed `price improvement` orders are immediately sent from Interactive Brokers to the BOX order book, and when the terms allow, IB will evaluate it for inclusion in a price improvement auction based on price and volume priority. In the auction, your order will have priority over broker-dealer price improvement orders at the same price. An Auction Limit order at a specified price. Use of a limit order ensures that you will not receive an execution at a price less favorable than the limit price. Enter limit orders in penny increments with your auction improvement amount computed as the difference between your limit order price and the nearest listed increment.

- Products: OPT (BOX only)

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "LMT";
            order.TotalQuantity = quantity;
            order.LmtPrice = price;
            order.AuctionStrategy = auctionStrategy;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("LMT");
        order.totalQuantity(quantity);
        order.lmtPrice(price);
        order.auctionStrategy(auctionStrategy);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LMT"
            order.TotalQuantity = quantity
            order.LmtPrice = price
            order.AuctionStrategy = auctionStrategy
```

```
    Order order;
    order.action = action;
    order.orderType = "LMT";
    order.totalQuantity = quantity;
    order.lmtPrice = price;
    order.auctionStrategy = auctionStrategy;
```

```
        order = Order()
        order.action = action
        order.orderType = "LMT"
        order.totalQuantity = quantity
        order.lmtPrice = price
        order.auctionStrategy = auctionStrategy
```

10.2.1.12 Auction Pegged to Stock

For option orders routed to the Boston Options Exchange (BOX) you may elect to participate in the BOX's price improvement auction in pennies. All BOX-directed `price improvement` orders are immediately sent from Interactive Brokers to the BOX order book, and when the terms allow, IB will evaluate it for inclusion in a price improvement auction based on price and volume priority. In the auction, your order will have priority over broker-dealer price improvement orders at the same price. An Auction Pegged to Stock order adjusts the order price by the product of a signed delta (which is entered as an absolute and assumed to be positive for calls, negative for puts) and the change of the option's underlying stock price. A buy or sell call order price is determined by adding the delta times a change in an underlying stock price change to a specified starting price for the call. To determine the change in price, a stock reference price (NBBO midpoint at the time of the order is assumed if no reference price is entered) is subtracted from the current NBBO midpoint. A stock range may also be entered that cancels an order when reached. The delta times the change in stock price will be rounded to the nearest penny in favor of the order and will be used as your auction improvement amount.

- Products: OPT (BOX only)

-
```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "PEG STK";
            order.TotalQuantity = quantity;
            order.Delta = delta;
            order.StartingPrice = startingPrice;
```

-
```
        Order order = new Order();
        order.action(action);
        order.orderType("PEG STK");
        order.totalQuantity(quantity);
        order.delta(delta);
        order.startingPrice(startingPrice);
```

-
```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "PEG STK"
            order.TotalQuantity = quantity
            order.Delta = delta
            order.StartingPrice = startingPrice
```

-
```
    Order order;
    order.action = action;
    order.orderType = "PEG STK";
    order.totalQuantity = quantity;
    order.delta = delta;
    order.startingPrice = startingPrice;
```

-
```
        order = Order()
        order.action = action
        order.orderType = "PEG STK"
        order.totalQuantity = quantity
        order.delta = delta
        order.startingPrice = startingPrice
```

10.2.1.13 Auction Relative

For option orders routed to the Boston Options Exchange (BOX) you may elect to participate in the BOX's price improvement auction in pennies. All BOX-directed `price improvement` orders are immediately sent from Interactive Brokers to the BOX order book, and when the terms allow, IB will evaluate it for inclusion in a price improvement auction based on price and volume priority. In the auction, your order will have priority over broker-dealer price improvement orders at the same price. An Auction Relative order that adjusts the order price by the product of a signed delta (which is entered as an absolute and assumed to be positive for calls, negative for puts) and the change of the option's underlying stock price. A buy or sell call order price is determined by adding the delta times a change in an underlying stock price change to a specified starting price for the call. To determine the change in price, a stock reference price (NBBO midpoint at the time of the order is assumed if no reference price is entered) is subtracted from the current NBBO midpoint. A stock range may also be entered that cancels an order when reached. The delta times the change in stock price will be rounded to the nearest penny in favor of the order and will be used as your auction improvement amount.

- Products: OPT (BOX only)

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "REL";
            order.TotalQuantity = quantity;
            order.AuxPrice = offset;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("REL");
        order.totalQuantity(quantity);
        order.auxPrice(offset);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "REL"
            order.TotalQuantity = quantity
            order.AuxPrice = offset
```

```
    Order order;
    order.action = action;
    order.orderType = "REL";
    order.totalQuantity = quantity;
    order.auxPrice = offset;
```

```
        order = Order()
        order.action = action
        order.orderType = "REL"
        order.totalQuantity = quantity
        order.auxPrice = offset
```

10.2.1.14 Block

The `Block` attribute is used for large volume option orders on ISE that consist of at least 50 contracts. To execute large-volume orders over time without moving the market, use the **Accumulate**/**Distribute** algorithm.

- Products: OPT
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "LMT";
            order.TotalQuantity = quantity;//Large volumes!
            order.LmtPrice = price;
            order.BlockOrder = true;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("LMT");
        order.totalQuantity(quantity);//Large volumes!
        order.lmtPrice(price);
        order.blockOrder(true);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LMT"
            order.TotalQuantity = quantity ' Large volumes!
            order.LmtPrice = price
            order.BlockOrder = True
```

```
    Order order;
    order.action = action;
    order.orderType = "LMT";
    order.totalQuantity = quantity;
    order.lmtPrice = price;
    order.blockOrder = true;
```

```
        order = Order()
        order.action = action
        order.orderType = "LMT"
        order.totalQuantity = quantity#Large volumes!
        order.lmtPrice = price
        order.blockOrder = True
```

10.2.1.15 Box Top

A `Box Top` order executes as a market order at the current best price. If the order is only partially filled, the remainder is submitted as a limit order with the limit price equal to the price at which the filled portion of the order executed.

- Products: OPT (BOX only)

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "BOX TOP";
            order.TotalQuantity = quantity;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("BOX TOP");
        order.totalQuantity(quantity);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "BOX TOP"
            order.TotalQuantity = quantity
```

```
    Order order;
    order.action = action;
    order.orderType = "BOX TOP";
    order.totalQuantity = quantity;
```

```
        order = Order()
        order.action = action
        order.orderType = "BOX TOP"
        order.totalQuantity = quantity
```

10.2.1.16 Limit Order

A `Limit order` is an order to buy or sell at a specified price or better. The Limit order ensures that if the order fills, it will not fill at a price less favorable than your limit price, but it does not guarantee a fill.

- Products: BOND, CFD, CASH, FUT, FOP, OPT, STK, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "LMT";
            order.TotalQuantity = quantity;
            order.LmtPrice = limitPrice;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("LMT");
        order.totalQuantity(quantity);
        order.lmtPrice(limitPrice);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LMT"
            order.TotalQuantity = quantity
            order.LmtPrice = limitPrice
```

```
    Order order;
    order.action = action;
    order.orderType = "LMT";
    order.totalQuantity = quantity;
    order.lmtPrice = limitPrice;
```

```
        order = Order()
        order.action = action
        order.orderType = "LMT"
        order.totalQuantity = quantity
        order.lmtPrice = limitPrice
```

10.2.1.17 Forex Cash Quantity Order

Forex orders can be placed in denomination of second currency in pair using cashQty field.
Requires TWS or IBG 963+

- Products: CASH
- `Forex Cash Quantity Orders`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "LMT";
            order.TotalQuantity = quantity;
            order.LmtPrice = limitPrice;
            order.CashQty = cashQty;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("LMT");
        order.totalQuantity(quantity);
        order.lmtPrice(limitPrice);
        order.cashQty(cashQty);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LMT"
            order.TotalQuantity = quantity
            order.LmtPrice = limitPrice
            order.CashQty = cashQty
```

```
    Order order;
    order.action = action;
    order.orderType = "LMT";
    order.totalQuantity = quantity;
    order.lmtPrice = limitPrice;
    order.cashQty = cashQty;
```

```
        order = Order()
        order.action = action
        order.orderType = "LMT"
        order.totalQuantity = quantity
        order.lmtPrice = limitPrice
        order.cashQty = cashQty
```

10.2.1.18 Limit if Touched

A `Limit if Touched` is an order to buy (or sell) a contract at a specified price or better, below (or above) the market. This order is held in the system until the trigger price is touched. An LIT order is similar to a stop limit order, except that an LIT sell order is placed above the current market price, and a stop limit sell order is placed below.

- Products: BOND, CFD, CASH, FUT, FOP, OPT, STK, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "LIT";
            order.TotalQuantity = quantity;
            order.LmtPrice = limitPrice;
            order.AuxPrice = triggerPrice;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("LIT");
        order.totalQuantity(quantity);
        order.lmtPrice(limitPrice);
        order.auxPrice(triggerPrice);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LIT"
            order.TotalQuantity = quantity
            order.LmtPrice = limitPrice
            order.AuxPrice = triggerPrice
```

```
    Order order;
    order.action = action;
    order.orderType = "LIT";
    order.totalQuantity = quantity;
    order.lmtPrice = limitPrice;
    order.auxPrice = triggerPrice;
```

```
        order = Order()
        order.action = action
        order.orderType = "LIT"
        order.totalQuantity = quantity
        order.lmtPrice = limitPrice
        order.auxPrice = triggerPrice
```

10.2.1.19 Limit on Close

A `Limit-on-close` (LOC) order will be submitted at the close and will execute if the closing price is at or better than the submitted limit price.

- Products: CFD, FUT, STK, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "LOC";
            order.TotalQuantity = quantity;
            order.LmtPrice = limitPrice;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("LOC");
        order.totalQuantity(quantity);
        order.lmtPrice(limitPrice);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LOC"
            order.TotalQuantity = quantity
            order.LmtPrice = limitPrice
```

```
    Order order;
    order.action = action;
    order.orderType = "LOC";
    order.totalQuantity = quantity;
    order.lmtPrice = limitPrice;
```

```
        order = Order()
        order.action = action
        order.orderType = "LOC"
        order.totalQuantity = quantity
        order.lmtPrice = limitPrice
```

10.2.1.20 Limit on Open

A `Limit-on-Open` (LOO) order combines a limit order with the OPG time in force to create an order that is submitted at the market's open, and that will only execute at the specified limit price or better. Orders are filled in accordance with specific exchange rules.

- Products: CFD, STK, OPT, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.Tif = "OPG";
            order.OrderType = "LMT";
            order.TotalQuantity = quantity;
            order.LmtPrice = limitPrice;
```

```
        Order order = new Order();
        order.action(action);
        order.tif("OPG");
        order.orderType("LOC");
        order.totalQuantity(quantity);
        order.lmtPrice(limitPrice);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.Tif = "OPG"
            order.OrderType = "LMT"
            order.TotalQuantity = quantity
            order.LmtPrice = limitPrice
```

```
    Order order;
    order.action = action;
    order.orderType = "LMT";
    order.tif = "OPG";
    order.totalQuantity = quantity;
    order.lmtPrice = limitPrice;
```

```
        order = Order()
        order.action = action
        order.tif = "OPG"
        order.orderType = "LMT"
        order.totalQuantity = quantity
        order.lmtPrice = limitPrice
```

10.2.1.21 Passive Relative

`Passive Relative` orders provide a means for traders to seek a less aggressive price than the National Best Bid and Offer (NBBO) while keeping the order pegged to the best bid (for a buy) or ask (for a sell). The order price is automatically adjusted as the markets move to keep the order less aggressive. For a buy order, your order price is pegged to the NBB by a less aggressive offset, and if the NBB moves up, your bid will also move up. If the NBB moves down, there will be no adjustment because your bid will become aggressive and execute. For a sell order, your price is pegged to the NBO by a less aggressive offset, and if the NBO moves down, your offer will also move down. If the NBO moves up, there will be no adjustment because your offer will become aggressive and execute. In addition to the offset, you can define an absolute cap, which works like a limit price, and will prevent your order from being executed above or below a specified level. The Passive Relative order is similar to the Relative/Pegged-to-Primary order, except that the Passive relative subtracts the offset from the bid and the Relative adds the offset to the bid.

- Products: STK, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "PASSV REL";
            order.TotalQuantity = quantity;
            order.AuxPrice = offset;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("PASSV REL");
        order.totalQuantity(quantity);
        order.auxPrice(offset);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "PASSV REL"
            order.TotalQuantity = quantity
            order.AuxPrice = offset
```

- ```
  Order order;
  order.action = action;
  order.orderType = "PASSV REL";
  order.totalQuantity = quantity;
  order.auxPrice = offset;
  ```

- ```
  order = Order()
  order.action = action
  order.orderType = "PASSV REL"
  order.totalQuantity = quantity
  order.auxPrice = offset
  ```

10.2.1.22 Pegged to Midpoint

A `pegged-to-midpoint` order provides a means for traders to seek a price at the midpoint of the National Best Bid and Offer (NBBO). The price automatically adjusts to peg the midpoint as the markets move, to remain aggressive. For a buy order, your bid is pegged to the NBBO midpoint and the order price adjusts automatically to continue to peg the midpoint if the market moves. The price only adjusts to be more aggressive. If the market moves in the opposite direction, the order will execute.

- Products: STK
- `Supported exchanges`

- ```
  Order order = new Order();
  order.Action = action;
  order.OrderType = "PEG MID";
  order.TotalQuantity = quantity;
  order.AuxPrice = offset;
  order.LmtPrice = limitPrice;
  ```

- ```
  Order order = new Order();
  order.action(action);
  order.orderType("PEG MID");
  order.totalQuantity(quantity);
  order.auxPrice(offset);
  order.lmtPrice(limitPrice);
  ```

- ```
  Dim order As Order = New Order
  order.Action = action
  order.OrderType = "PEG MID"
  order.TotalQuantity = quantity
  order.AuxPrice = offset
  order.LmtPrice = limitPrice
  ```

- ```
  Order order;
  order.action = action;
  order.orderType = "PEG MID";
  order.totalQuantity = quantity;
  order.auxPrice = offset;
  order.lmtPrice = limitPrice;
  ```

- ```
  order = Order()
  order.action = action
  order.orderType = "PEG MID"
  order.totalQuantity = quantity
  order.auxPrice = offset
  order.lmtPrice = limitPrice
  ```

10.2.1.23 Market to Limit

A `Market-to-Limit` (MTL) order is submitted as a market order to execute at the current best market price. If the order is only partially filled, the remainder of the order is canceled and re-submitted as a limit order with the limit price equal to the price at which the filled portion of the order executed.

- Products: CFD, FUT, FOP, OPT, STK, WAR
- `Supported exchanges`
  ```

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "MTL";
            order.TotalQuantity = quantity;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("MTL");
        order.totalQuantity(quantity);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "MTL"
            order.TotalQuantity = quantity
```

```
    Order order;
    order.action = action;
    order.orderType = "MTL";
    order.totalQuantity = quantity;
```

```
        order = Order()
        order.action = action
        order.orderType = "MTL"
        order.totalQuantity = quantity
```

10.2.1.24 Market with Protection

This order type is useful for futures traders using Globex. A `Market with Protection` order is a market order that will be cancelled and resubmitted as a limit order if the entire order does not immediately execute at the market price. The limit price is set by Globex to be close to the current market price, slightly higher for a sell order and lower for a buy order.

- Products: FUT, FOP
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "MKT PRT";
            order.TotalQuantity = quantity;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("MKT PRT");
        order.totalQuantity(quantity);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "MKT PRT"
            order.TotalQuantity = quantity
```

```
    Order order;
    order.action = action;
    order.orderType = "MKT PRT";
    order.totalQuantity = quantity;
```

```
        order = Order()
        order.action = action
        order.orderType = "MKT PRT"
        order.totalQuantity = quantity
```

10.2.1.25 Stop

A `Stop` order is an instruction to submit a buy or sell market order if and when the user-specified stop trigger price is attained or penetrated. A Stop order is not guaranteed a specific execution price and may execute significantly away from its stop price. A Sell Stop order is always placed below the current market price and is typically used to limit a loss or protect a profit on a long stock position. A Buy Stop order is always placed above the current market price. It is typically used to limit a loss or help protect a profit on a short sale.

- Products: CFD, BAG, CASH, FUT, FOP, OPT, STK, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "STP";
            order.AuxPrice = stopPrice;
            order.TotalQuantity = quantity;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("STP");
        order.auxPrice(stopPrice);
        order.totalQuantity(quantity);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "STP"
            order.AuxPrice = stopPrice
            order.TotalQuantity = quantity
```

```
    Order order;
    order.action = action;
    order.orderType = "STP";
    order.totalQuantity = quantity;
    order.auxPrice = stopPrice;
```

```
        order = Order()
        order.action = action
        order.orderType = "STP"
        order.auxPrice = stopPrice
        order.totalQuantity = quantity
```

10.2.1.26 Stop Limit

A `Stop-Limit` order is an instruction to submit a buy or sell limit order when the user-specified stop trigger price is attained or penetrated. The order has two basic components: the stop price and the limit price. When a trade has occurred at or through the stop price, the order becomes executable and enters the market as a limit order, which is an order to buy or sell at a specified price or better.

- Products: CFD, CASH, FUT, FOP, OPT, STK, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "STP LMT";
            order.TotalQuantity = quantity;
            order.LmtPrice = limitPrice;
            order.AuxPrice = stopPrice;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("STP LMT");
        order.lmtPrice(limitPrice);
        order.auxPrice(stopPrice);
        order.totalQuantity(quantity);
```

```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "STP LMT"
            order.TotalQuantity = quantity
            order.LmtPrice = limitPrice
            order.AuxPrice = stopPrice
```

```
    Order order;
    order.action = action;
    order.orderType = "STP LMT";
    order.totalQuantity = quantity;
    order.lmtPrice = limitPrice;
    order.auxPrice = stopPrice;
```

```
        order = Order()
        order.action = action
        order.orderType = "STP LMT"
        order.totalQuantity = quantity
        order.lmtPrice = limitPrice
        order.auxPrice = stopPrice
```

10.2.1.27 Stop with Protection

A `Stop with Protection` order combines the functionality of a stop limit order with a market with protection order. The order is set to trigger at a specified stop price. When the stop price is penetrated, the order is triggered as a market with protection order, which means that it will fill within a specified protected price range equal to the trigger price +/- the exchange-defined protection point range. Any portion of the order that does not fill within this protected range is submitted as a limit order at the exchange-defined trigger price +/- the protection points.

- Products: FUT
- `Supported exchanges`

```
            Order order = new Order();
            order.TotalQuantity = quantity;
            order.Action = action;
            order.OrderType = "STP PRT";
            order.AuxPrice = stopPrice;
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("STP PRT");
        order.auxPrice(stopPrice);
        order.totalQuantity(quantity);
```

```
            Dim order As Order = New Order
            order.TotalQuantity = quantity
            order.Action = action
            order.OrderType = "STP PRT"
            order.AuxPrice = stopPrice
```

```
    Order order;
    order.action = action;
    order.orderType = "STP PRT";
    order.totalQuantity = quantity;
    order.auxPrice = stopPrice;
```

```
        order = Order()
        order.totalQuantity = quantity
        order.action = action
        order.orderType = "STP PRT"
        order.auxPrice = stopPrice
```

10.2.1.28 Trailing Stop

A sell `trailing stop` order sets the stop price at a fixed amount below the market price with an attached "trailing" amount. As the market price rises, the stop price rises by the trail amount, but if the stock price falls, the stop loss price doesn't change, and a market order is submitted when the stop price is hit. This technique is designed to allow an investor to specify a limit on the maximum possible loss, without setting a limit on the maximum possible gain. "Buy" trailing stop orders are the mirror image of sell trailing stop orders, and are most appropriate for use in falling markets.

Note that Trailing Stop orders can have the trailing amount specified as a percent, as in the example below, or as an absolute amount which is specified in the auxPrice field.

- Products: CFD, CASH, FOP, FUT, OPT, STK, WAR
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "TRAIL";
            order.TotalQuantity = quantity;
            order.TrailingPercent = trailingPercent;
            order.TrailStopPrice = trailStopPrice;
```

- ```
        Order order = new Order();
        order.action(action);
        order.orderType("TRAIL");
        order.trailingPercent(trailingPercent);
        order.trailStopPrice(trailStopPrice);
        order.totalQuantity(quantity);
  ```

- ```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "TRAIL"
            order.TotalQuantity = quantity
            order.TrailingPercent = trailingPercent
            order.TrailStopPrice = trailStopPrice
  ```

- ```
    Order order;
    order.action = action;
    order.orderType = "TRAIL";
    order.totalQuantity = quantity;
    order.trailingPercent = trailingPercent;
    order.trailStopPrice = trailStopPrice;
  ```

- ```
        order = Order()
        order.action = action
        order.orderType = "TRAIL"
        order.totalQuantity = quantity
        order.trailingPercent = trailingPercent
        order.trailStopPrice = trailStopPrice
  ```

10.2.1.29 Trailing Stop Limit

A `trailing stop limit` order is designed to allow an investor to specify a limit on the maximum possible loss, without setting a limit on the maximum possible gain. A SELL trailing stop limit moves with the market price, and continually recalculates the stop trigger price at a fixed amount below the market price, based on the user-defined "trailing" amount. The limit order price is also continually recalculated based on the limit offset. As the market price rises, both the stop price and the limit price rise by the trail amount and limit offset respectively, but if the stock price falls, the stop price remains unchanged, and when the stop price is hit a limit order is submitted at the last calculated limit price. A "Buy" trailing stop limit order is the mirror image of a sell trailing stop limit, and is generally used in falling markets.

Trailing Stop Limit orders can be sent with the trailing amount specified as an absolute amount, as in the example below, or as a percentage, specified in the trailingPercent field.

Important: the 'limit offset' field is set by default in the TWS/IBG settings in v963+. This setting either needs to be changed in the Order Presets, the default value accepted, or the limit price offset sent from the API as in the example below. Not both the 'limit price' and 'limit price offset' fields can be set in TRAIL LIMIT orders.

- Products: BOND, CFD, CASH, FUT, FOP, OPT, STK, WAR

- ```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "TRAIL LIMIT";
            order.TotalQuantity = quantity;
            order.TrailStopPrice = trailStopPrice;
            order.LmtPriceOffset = lmtPriceOffset;
            order.AuxPrice = trailingAmount;
  ```

- ```
        Order order = new Order();
        order.action(action);
        order.orderType("TRAIL LIMIT");
        order.lmtPriceOffset(lmtPriceOffset);
        order.auxPrice(trailingAmount);
        order.trailStopPrice(trailStopPrice);
        order.totalQuantity(quantity);
  ```

- ```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "TRAIL LIMIT"
            order.TotalQuantity = quantity
            order.LmtPriceOffset = lmtPriceOffset
            order.TrailStopPrice = trailStopPrice
            order.AuxPrice = trailingAmount
  ```
  ```

- ```
  Order order;
  order.action = action;
  order.orderType = "TRAIL LIMIT";
  order.totalQuantity = quantity;
  order.trailStopPrice = trailStopPrice;
  order.lmtPriceOffset = lmtPriceOffset;
  order.auxPrice = trailingAmount;
  ```

- ```
      order = Order()
      order.action = action
      order.orderType = "TRAIL LIMIT"
      order.totalQuantity = quantity
      order.trailStopPrice = trailStopPrice
      order.lmtPriceOffset = lmtPriceOffset
      order.auxPrice = trailingAmount
  ```

10.2.1.30 Combo Limit

Create combination orders that include options, stock and futures legs (stock legs can be included if the order is routed through SmartRouting). Although a combination/spread order is constructed of separate legs, it is executed as a single transaction if it is routed directly to an exchange. For combination orders that are SmartRouted, each leg may be executed separately to ensure best execution.

- Products: OPT, STK, FUT

- ```
          Order order = new Order();
          order.Action = action;
          order.OrderType = "LMT";
          order.TotalQuantity = quantity;
          order.LmtPrice = limitPrice;
          if (nonGuaranteed)
          {
              order.SmartComboRoutingParams = new List<TagValue>();
              order.SmartComboRoutingParams.Add(new TagValue("NonGuaranteed", "1"));
          }
  ```

- ```
      Order order = new Order();
      order.action(action);
      order.orderType("LMT");
      order.lmtPrice(limitPrice);
      order.totalQuantity(quantity);
      if (nonGuaranteed)
      {
          order.smartComboRoutingParams().add(new TagValue("NonGuaranteed", "1"));
      }
  ```

- ```
          Dim order As Order = New Order
          order.Action = action
          order.OrderType = "LMT"
          order.TotalQuantity = quantity
          order.LmtPrice = limitPrice
          If (nonGuaranteed) Then
              order.SmartComboRoutingParams = New List(Of TagValue)
              order.SmartComboRoutingParams.Add(New TagValue("NonGuaranteed", "1"))
          End If
  ```

- ```
  Order order;
  order.action = action;
  order.orderType = "LMT";
  order.totalQuantity = quantity;
  order.lmtPrice = limitPrice;
  if(nonGuaranteed){
      TagValueSPtr tag1(new TagValue("NonGuaranteed", "1"));
      order.smartComboRoutingParams.reset(new TagValueList());
      order.smartComboRoutingParams->push_back(tag1);
  }
  ```

- ```
      order = Order()
      order.action = action
      order.orderType = "LMT"
      order.totalQuantity = quantity
      order.lmtPrice = limitPrice
      if nonGuaranteed:

          order.smartComboRoutingParams = []
          order.smartComboRoutingParams.append(TagValue("NonGuaranteed", "1"))
  ```
  ```

10.2.1.31 Combo Market

Create combination orders that include options, stock and futures legs (stock legs can be included if the order is routed through SmartRouting). Although a combination/spread order is constructed of separate legs, it is executed as a single transaction if it is routed directly to an exchange. For combination orders that are SmartRouted, each leg may be executed separately to ensure best execution.

- Products: OPT, STK, FUT

- ```
  Order order = new Order();
  order.Action = action;
  order.OrderType = "MKT";
  order.TotalQuantity = quantity;
  if (nonGuaranteed)
  {
      order.SmartComboRoutingParams = new List<TagValue>();
      order.SmartComboRoutingParams.Add(new TagValue("NonGuaranteed", "1"));
  }
  ```

- ```
  Order order = new Order();
  order.action(action);
  order.orderType("MKT");
  order.totalQuantity(quantity);
  if (nonGuaranteed)
  {
      order.smartComboRoutingParams().add(new TagValue("NonGuaranteed", "1"));
  }
  ```

- ```
  Dim order As Order = New Order
  order.Action = action
  order.OrderType = "MKT"
  order.TotalQuantity = quantity
  If (nonGuaranteed) Then

      order.SmartComboRoutingParams = New List(Of TagValue)
      order.SmartComboRoutingParams.Add(New TagValue("NonGuaranteed", "1"))
  End If
  ```

- ```
  Order order;
  order.action = action;
  order.orderType = "MKT";
  order.totalQuantity = quantity;
  if(nonGuaranteed){
      TagValueSPtr tag1(new TagValue("NonGuaranteed", "1"));
      order.smartComboRoutingParams.reset(new TagValueList());
      order.smartComboRoutingParams->push_back(tag1);
  }
  ```

- ```
  order = Order()
  order.action = action
  order.orderType = "MKT"
  order.totalQuantity = quantity
  if nonGuaranteed:

      order.smartComboRoutingParams = []
      order.smartComboRoutingParams.append(TagValue("NonGuaranteed", "1"))
  ```

10.2.1.32 Combo Limit with Price per Leg

Create combination orders that include options, stock and futures legs (stock legs can be included if the order is routed through SmartRouting). Although a combination/spread order is constructed of separate legs, it is executed as a single transaction if it is routed directly to an exchange. For combination orders that are SmartRouted, each leg may be executed separately to ensure best execution.

- Products: OPT, STK, FUT
  ```

- ```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "LMT";
            order.TotalQuantity = quantity;
            order.OrderComboLegs = new List<OrderComboLeg>();
            foreach(double price in legPrices)
            {
                OrderComboLeg comboLeg = new OrderComboLeg();
                comboLeg.Price = 5.0;
                order.OrderComboLegs.Add(comboLeg);
            }
            if (nonGuaranteed)
            {
                order.SmartComboRoutingParams = new List<TagValue>();
                order.SmartComboRoutingParams.Add(new TagValue("NonGuaranteed", "1"));
            }
  ```

- ```
        Order order = new Order();
        order.action(action);
        order.orderType("LMT");
        order.totalQuantity(quantity);
        order.orderComboLegs(new ArrayList<>());

        for(double price : legPrices) {
            OrderComboLeg comboLeg = new OrderComboLeg();
            comboLeg.price(5.0);
            order.orderComboLegs().add(comboLeg);
        }

        if (nonGuaranteed)
        {
            order.smartComboRoutingParams().add(new TagValue("NonGuaranteed", "1"));
        }
  ```

- ```
            Dim order As Order = New Order
            order.Action = action
            order.OrderType = "LMT"
            order.TotalQuantity = quantity
            order.OrderComboLegs = New List(Of OrderComboLeg)
            For Each price As Double In legPrices
                Dim ComboLeg As OrderComboLeg = New OrderComboLeg
                ComboLeg.Price = 5.0
                order.OrderComboLegs.Add(ComboLeg)
            Next price

            If (nonGuaranteed) Then
                order.SmartComboRoutingParams = New List(Of TagValue)
                order.SmartComboRoutingParams.Add(New TagValue("NonGuaranteed", "1"))
            End If
  ```

- ```
    Order order;
    order.action = action;
    order.orderType = "LMT";
    order.totalQuantity = quantity;
    order.orderComboLegs.reset(new Order::OrderComboLegList());
    for(unsigned int i = 0; i < legprices.size(); i++){
        OrderComboLegSPtr comboLeg(new OrderComboLeg());
        comboLeg->price = legprices[i];
        order.orderComboLegs->push_back(comboLeg);
    }
    if(nonGuaranteed){
        TagValueSPtr tag1(new TagValue("NonGuaranteed", "1"));
        order.smartComboRoutingParams.reset(new TagValueList());
        order.smartComboRoutingParams->push_back(tag1);
    }
  ```

- ```
        order = Order()
        order.action = action
        order.orderType = "LMT"
        order.totalQuantity = quantity
        order.orderComboLegs = []
        for price in legPrices:

            comboLeg = OrderComboLeg()
            comboLeg.price = price
            order.orderComboLegs.append(comboLeg)

        if nonGuaranteed:
            order.smartComboRoutingParams = []
            order.smartComboRoutingParams.append(TagValue("NonGuaranteed", "1"))
  ```
  ```

10.2.1.33 Relative Limit Combo

Create combination orders that include options, stock and futures legs (stock legs can be included if the order is routed through SmartRouting). Although a combination/spread order is constructed of separate legs, it is executed as a single transaction if it is routed directly to an exchange. For combination orders that are SmartRouted, each leg may be executed separately to ensure best execution.

- Products: OPT, STK, FUT

- ```
  Order order = new Order();
  order.Action = action;
  order.TotalQuantity = quantity;
  order.OrderType = "REL + LMT";
  order.LmtPrice = limitPrice;
  if (nonGuaranteed)
  {
      order.SmartComboRoutingParams = new List<TagValue>();
      order.SmartComboRoutingParams.Add(new TagValue("NonGuaranteed", "1"));
  }
  ```

- ```
  Order order = new Order();
  order.action(action);
  order.orderType("REL + LMT");
  order.totalQuantity(quantity);
  order.lmtPrice(limitPrice);

  if (nonGuaranteed)
  {
      order.smartComboRoutingParams().add(new TagValue("NonGuaranteed", "1"));
  }
  ```

- ```
  Dim order As Order = New Order
  order.Action = action
  order.TotalQuantity = quantity
  order.OrderType = "REL + LMT"
  order.LmtPrice = limitPrice
  If (nonGuaranteed) Then
      order.SmartComboRoutingParams = New List(Of TagValue)
      order.SmartComboRoutingParams.Add(New TagValue("NonGuaranteed", "1"))
  End If
  ```

- ```
  Order order;
  order.action = action;
  order.totalQuantity = quantity;
  order.orderType = "Rel + LMT";
  order.lmtPrice = limitPrice;
  if(nonGuaranteed){
      TagValueSPtr tag1(new TagValue("NonGuaranteed", "1"));
      order.smartComboRoutingParams.reset(new TagValueList());
      order.smartComboRoutingParams->push_back(tag1);
  }
  ```

- ```
  order = Order()
  order.action = action
  order.totalQuantity = quantity
  order.orderType = "REL + LMT"
  order.lmtPrice = limitPrice
  if nonGuaranteed:

      order.smartComboRoutingParams = []
      order.smartComboRoutingParams.append(TagValue("NonGuaranteed", "1"))
  ```

10.2.1.34 Relative Market Combo

Create combination orders that include options, stock and futures legs (stock legs can be included if the order is routed through SmartRouting). Although a combination/spread order is constructed of separate legs, it is executed as a single transaction if it is routed directly to an exchange. For combination orders that are SmartRouted, each leg may be executed separately to ensure best execution.

- Products: OPT, STK, FUT
  ```

```
            Order order = new Order();
            order.Action = action;
            order.TotalQuantity = quantity;
            order.OrderType = "REL + MKT";
            if (nonGuaranteed)
            {
                order.SmartComboRoutingParams = new List<TagValue>();
                order.SmartComboRoutingParams.Add(new TagValue("NonGuaranteed", "1"));
            }
```

```
        Order order = new Order();
        order.action(action);
        order.orderType("REL + MKT");
        order.totalQuantity(quantity);
        if (nonGuaranteed)
        {
            order.smartComboRoutingParams().add(new TagValue("NonGuaranteed", "1"));
        }
```

```
            Dim order As Order = New Order
            order.Action = action
            order.TotalQuantity = quantity
            order.OrderType = "REL + MKT"
            If (nonGuaranteed) Then

                order.SmartComboRoutingParams = New List(Of TagValue)
                order.SmartComboRoutingParams.Add(New TagValue("NonGuaranteed", "1"))
            End If
```

```
    Order order;
    order.action = action;
    order.totalQuantity = quantity;
    order.orderType = "Rel + MKT";
    if(nonGuaranteed){
        TagValueSPtr tag1(new TagValue("NonGuaranteed", "1"));
        order.smartComboRoutingParams.reset(new TagValueList());
        order.smartComboRoutingParams->push_back(tag1);
    }
```

```
        order = Order()
        order.action = action
        order.totalQuantity = quantity
        order.orderType = "REL + MKT"
        if nonGuaranteed:

            order.smartComboRoutingParams = []
            order.smartComboRoutingParams.append(TagValue("NonGuaranteed", "1"))
```

10.2.1.35 Volatility

Specific to US options, investors are able to create and enter `Volatility-type` orders for options and combinations rather than price orders. Option traders may wish to trade and position for movements in the price of the option determined by its implied volatility. Because implied volatility is a key determinant of the premium on an option, traders position in specific contract months in an effort to take advantage of perceived changes in implied volatility arising before, during or after earnings or when company specific or broad market volatility is predicted to change. In order to create a Volatility order, clients must first create a Volatility Trader page from the Trading Tools menu and as they enter option contracts, premiums will display in percentage terms rather than premium. The buy/sell process is the same as for regular orders priced in premium terms except that the client can limit the volatility level they are willing to pay or receive.

- Products: FOP, OPT
- `Supported exchanges`

```
            Order order = new Order();
            order.Action = action;
            order.OrderType = "VOL";
            order.TotalQuantity = quantity;
            order.Volatility = volatilityPercent;//Expressed in percentage (40%)
            order.VolatilityType = volatilityType;// 1=daily, 2=annual
```

```
Order order = new Order();
order.action(action);
order.orderType("VOL");
order.volatility(volatilityPercent);//Expressed in percentage (40%)
order.volatilityType(volatilityType);// 1=daily, 2=annual
order.totalQuantity(quantity);
```

```
    Dim order As Order = New Order
    order.Action = action
    order.OrderType = "VOL"
    order.TotalQuantity = quantity
    order.Volatility = volatilityPercent  'Expressed in percentage (40%)
    order.VolatilityType = volatilityType  ' 1=daily, 2=annual
```

```
Order order;
order.action = action;
order.orderType = "VOL";
order.totalQuantity = quantity;
order.volatility = volatilityPercent; //Expressed in percentage (40%)
order.volatilityType = volatilityType; // 1=daily, 2=annual
```

```
order = Order()
order.action = action
order.orderType = "VOL"
order.totalQuantity = quantity
order.volatility = volatilityPercent#Expressed in percentage (40%)
order.volatilityType = volatilityType# 1=daily, 2=annual
```

10.2.1.36 Pegged to Benchmark

The `Pegged to Benchmark` order is similar to the Pegged to Stock order for options, except that the Pegged to Benchmark allows you to specify any asset type as the reference (benchmark) contract for a stock or option order. Both the primary and reference contracts must use the same currency.

- Products: STK, OPT
- `Supported exchanges`

```
    Order order = new Order();
    order.OrderType = "PEG BENCH";
    //BUY or SELL
    order.Action = action;
    order.TotalQuantity = quantity;
    //Beginning with price...
    order.StartingPrice = startingPrice;
    //increase/decrease price..
    order.IsPeggedChangeAmountDecrease = peggedChangeAmountDecrease;
    //by... (and likewise for price moving in opposite direction)
    order.PeggedChangeAmount = peggedChangeAmount;
    //whenever there is a price change of...
    order.ReferenceChangeAmount = referenceChangeAmount;
    //in the reference contract...
    order.ReferenceContractId = referenceConId;
    //being traded at...
    order.ReferenceExchange = referenceExchange;
    //starting reference price is...
    order.StockRefPrice = stockReferencePrice;
    //Keep order active as long as reference contract trades between...
    order.StockRangeLower = referenceContractLowerRange;
    //and...
    order.StockRangeUpper = referenceContractUpperRange;
```

```
Order order = new Order();
order.orderType("PEG BENCH");
//BUY or SELL
order.action(action);
order.totalQuantity(quantity);
//Beginning with price...
order.startingPrice(startingPrice);
//increase/decrease price...
order.isPeggedChangeAmountDecrease(peggedChangeAmountDecrease);
//by... (and likewise for price moving in opposite direction)
order.peggedChangeAmount(peggedChangeAmount);
//whenever there is a price change of...
order.referenceChangeAmount(referenceChangeAmount);
//in the reference contract...
order.referenceContractId(referenceConId);
//being traded at...
```

```
order.referenceExchangeId(referenceExchange);
//starting reference price is...
order.stockRefPrice(stockReferencePrice);
//Keep order active as long as reference contract trades between...
order.stockRangeLower(referenceContractLowerRange);
//and...
order.stockRangeUpper(referenceContractUpperRange);
```

- ```
  Dim order As Order = New Order
  order.OrderType = "PEG BENCH"
  'BUY Or SELL
  order.Action = action
  order.TotalQuantity = quantity
  'Beginning with price...
  order.StartingPrice = startingPrice
  'increase/decrease price..
  order.IsPeggedChangeAmountDecrease = peggedChangeAmountDecrease
  'by... (And likewise for price moving in opposite direction)
  order.PeggedChangeAmount = peggedChangeAmount
  'whenever there Is a price change of...
  order.ReferenceChangeAmount = referenceChangeAmount
  'in the reference contract...
  order.ReferenceContractId = referenceConId
  'being traded at...
  order.ReferenceExchange = referenceExchange
  'starting reference price Is...
  order.StockRefPrice = stockReferencePrice
  'Keep order active as long as reference contract trades between...
  order.StockRangeLower = referenceContractLowerRange
  'And...
  order.StockRangeUpper = referenceContractUpperRange
  ```

- ```
  Order order;
  order.orderType = "PEG BENCH";
  //BUY or SELL
  order.action = action;
  order.totalQuantity = quantity;
  //Beginning with price...
  order.startingPrice = startingPrice;
  //increase/decrease price..
  order.isPeggedChangeAmountDecrease = peggedChangeAmountDecrease;
  //by... (and likewise for price moving in opposite direction)
  order.peggedChangeAmount = peggedChangeAmount;
  //whenever there is a price change of...
  order.referenceChangeAmount = referenceChangeAmount;
  //in the reference contract...
  order.referenceContractId = referenceConId;
  //being traded at...
  order.referenceExchangeId = referenceExchange;
  //starting reference price is...
  order.stockRefPrice = stockReferencePrice;
  //Keep order active as long as reference contract trades between...
  order.stockRangeLower = referenceContractLowerRange;
  //and...
  order.stockRangeUpper = referenceContractUpperRange;
  ```

- ```
  order = Order()
  order.orderType = "PEG BENCH"
  #BUY or SELL
  order.action = action
  order.totalQuantity = quantity
  #Beginning with price...
  order.startingPrice = startingPrice
  #increase/decrease price..
  order.isPeggedChangeAmountDecrease = peggedChangeAmountDecrease
  #by... (and likewise for price moving in opposite direction)
  order.peggedChangeAmount = peggedChangeAmount
  #whenever there is a price change of...
  order.referenceChangeAmount = referenceChangeAmount
  #in the reference contract...
  order.referenceContractId = referenceConId
  #being traded at...
  order.referenceExchange = referenceExchange
  #starting reference price is...
  order.stockRefPrice = stockReferencePrice
  #Keep order active as long as reference contract trades between...
  order.stockRangeLower = referenceContractLowerRange
  #and...
  order.stockRangeUpper = referenceContractUpperRange
  ```
  ```

10.2.2 Advanced Orders and Algos

On top of the basic order types, it is possible to make use of the following advanced features:

- **Hedging**
- **Bracket Orders**
- **One Cancels All**
- **Adjustable stops**
- **Conditioning**
- **IB Algos**

10.2.3 Hedging

Hedging orders are similar to Bracket Orders. With hedging order, a child order is submitted only on execution of the parent. Orders can be hedged by an attached forex trade, Beta hedge, or Pair Trade, just as in TWS:
`Hedging Orders in TWS`

For an example of a forex hedge, when buying a contract on a currency other than your base, you can attach an FX order to convert base currency to the currency of the contract to cover the cost of the trade thanks to the TWS API's **Attaching Orders** mechanism.

- ```
  public static Order MarketFHedge(int parentOrderId, string action)
  {
      //FX Hedge orders can only have a quantity of 0
      Order order = MarketOrder(action, 0);
      order.ParentId = parentOrderId;
      order.HedgeType = "F";
      return order;
  }
  ```

  ...

  ```
      //Parent order on a contract which currency differs from your base currency
      Order parent = OrderSamples.LimitOrder("BUY", 100, 10);
      parent.OrderId = nextOrderId++;
      //Hedge on the currency conversion
      Order hedge = OrderSamples.MarketFHedge(parent.OrderId, "BUY");
      //Place the parent first...
      client.placeOrder(parent.OrderId, ContractSamples.EuropeanStock(), parent);
      //Then the hedge order
      client.placeOrder(nextOrderId++, ContractSamples.EurGbpFx(), hedge);
  ```

- ```
  public static Order MarketFHedge(int parentOrderId, String action) {
      //FX Hedge orders can only have a quantity of 0
      Order order = MarketOrder(action, 0);
      order.parentId(parentOrderId);
      order.hedgeType("F");
      return order;
  }
  ```

  ...

  ```
      //Parent order on a contract which currency differs from your base currency
      Order parent = OrderSamples.LimitOrder("BUY", 100, 10);
      parent.orderId(nextOrderId++);
      //Hedge on the currency conversion
      Order hedge = OrderSamples.MarketFHedge(parent.orderId(), "BUY");
      //Place the parent first...
      client.placeOrder(parent.orderId(), ContractSamples.EuropeanStock(), parent);
      //Then the hedge order
      client.placeOrder(nextOrderId++, ContractSamples.EurGbpFx(), hedge);
  ```

-
```
        Public Shared Function MarketFHedge(parentOrderId As Integer, action As String) As Order

            'FX Hedge orders can only have a quantity of 0
            Dim Order As Order = MarketOrder(action, 0)
            Order.ParentId = parentOrderId
            Order.HedgeType = "F"
            Return Order
        End Function
```

...

```
        'Parent order on a contract which currency differs from your base currency
        Dim parent As Order = OrderSamples.LimitOrder("BUY", 100, 10)
        parent.OrderId = increment(nextOrderId)
        'Hedge on the currency conversion
        Dim hedge As Order = OrderSamples.MarketFHedge(parent.OrderId, "BUY")
        'Place the parent first...
        client.placeOrder(parent.OrderId, ContractSamples.EuropeanStock(), parent)
        'Then the hedge order
        client.placeOrder(increment(increment(nextOrderId)), ContractSamples.EurGbpFx(), hedge)
```

-
```
Order OrderSamples::MarketFHedge(int parentOrderId, std::string action){
    //FX Hedge orders can only have a quantity of 0
    Order order = MarketOrder(action, 0);
    order.parentId = parentOrderId;
    order.hedgeType = "F";
    return order;
}
```

...

```
    //Parent order on a contract which currency differs from your base currency
    Order parent = OrderSamples::LimitOrder("BUY", 100, 10);
    parent.orderId = m_orderId++;
    //Hedge on the currency conversion
    Order hedge = OrderSamples::MarketFHedge(parent.orderId, "BUY");
    //Place the parent first...
    m_pClient->placeOrder(parent.orderId, ContractSamples::EuropeanStock(), parent);
    //Then the hedge order
    m_pClient->placeOrder(m_orderId++, ContractSamples::EurGbpFx(), hedge);
```

-
```
    @staticmethod
    def MarketFHedge(parentOrderId:int, action:str):

        #FX Hedge orders can only have a quantity of 0
        order = OrderSamples.MarketOrder(action, 0)
        order.parentId = parentOrderId
        order.hedgeType = "F"
        return order
```

...

```
        # Parent order on a contract which currency differs from your base currency
        parent = OrderSamples.LimitOrder("BUY", 100, 10)
        parent.orderId = self.nextOrderId()
        # Hedge on the currency conversion
        hedge = OrderSamples.MarketFHedge(parent.orderId, "BUY")
        # Place the parent first...
        self.placeOrder(parent.orderId, ContractSamples.EuropeanStock(), parent)
        # Then the hedge order
        self.placeOrder(self.nextOrderId(), ContractSamples.EurGbpFx(), hedge)
```

Note that in some cases it will be necessary to include a small delay of 50 ms or less after placing the parent order for processing, before placing the child order. Otherwise the error "10006: Missing parent order" will be triggered.

10.2.4 Bracket Orders

`Bracket Orders` are designed to help limit your loss and lock in a profit by "bracketing" an order with two opposite-side orders. A BUY order is bracketed by a high-side sell limit order and a low-side sell stop order. A SELL

order is bracketed by a high-side buy stop order and a low side buy limit order. Note how bracket orders make use of the TWS API's **Attaching Orders** mechanism.

One key thing to keep in mind is to handle the order transmission accurately. Since a Bracket consists of three orders, there is always a risk that at least one of the orders gets filled before the entire bracket is sent. To avoid it, make use of the **IBApi.Order.Transmit** flag. When this flag is set to 'false', the TWS will receive the order but it will not send (transmit) it to the servers. In the example below, the first (parent) and second (takeProfit) orders will be send to the TWS but not transmitted to the servers. When the last child order (stopLoss) is sent however and given that its **IBApi.Order.Transmit** flag is set to true, the TWS will interpret this as a signal to transmit not only its parent order but also the rest of siblings, removing the risks of an accidental execution.

- ```
  public static List<Order> BracketOrder(int parentOrderId, string action, double quantity, double limitPrice,
        double takeProfitLimitPrice, double stopLossPrice)
    {
        //This will be our main or "parent" order
        Order parent = new Order();
        parent.OrderId = parentOrderId;
        parent.Action = action;
        parent.OrderType = "LMT";
        parent.TotalQuantity = quantity;
        parent.LmtPrice = limitPrice;
        //The parent and children orders will need this attribute set to false to prevent accidental executions.
        //The LAST CHILD will have it set to true,
        parent.Transmit = false;

        Order takeProfit = new Order();
        takeProfit.OrderId = parent.OrderId + 1;
        takeProfit.Action = action.Equals("BUY") ? "SELL" : "BUY";
        takeProfit.OrderType = "LMT";
        takeProfit.TotalQuantity = quantity;
        takeProfit.LmtPrice = takeProfitLimitPrice;
        takeProfit.ParentId = parentOrderId;
        takeProfit.Transmit = false;

        Order stopLoss = new Order();
        stopLoss.OrderId = parent.OrderId + 2;
        stopLoss.Action = action.Equals("BUY") ? "SELL" : "BUY";
        stopLoss.OrderType = "STP";
        //Stop trigger price
        stopLoss.AuxPrice = stopLossPrice;
        stopLoss.TotalQuantity = quantity;
        stopLoss.ParentId = parentOrderId;
        //In this case, the low side order will be the last child being sent. Therefore, it needs to set this attribute to true
        //to activate all its predecessors
        stopLoss.Transmit = true;

        List<Order> bracketOrder = new List<Order>();
        bracketOrder.Add(parent);
        bracketOrder.Add(takeProfit);
        bracketOrder.Add(stopLoss);
        return bracketOrder;
    }
  ```

...

```
        List<Order> bracket = OrderSamples.BracketOrder(nextOrderId++, "BUY", 100, 30, 40, 20);
        foreach (Order o in bracket)
            client.placeOrder(o.OrderId, ContractSamples.EuropeanStock(), o);
```

- ```
  public static List<Order> BracketOrder(int parentOrderId, String action, double quantity, double limitPrice, double takeProfitLimitPrice, double stopLossPrice) {
    //This will be our main or "parent" order
    Order parent = new Order();
    parent.orderId(parentOrderId);
    parent.action(action);
    parent.orderType("LMT");
    parent.totalQuantity(quantity);
    parent.lmtPrice(limitPrice);
    //The parent and children orders will need this attribute set to false to prevent accidental executions.
    //The LAST CHILD will have it set to true.
    parent.transmit(false);

    Order takeProfit = new Order();
    takeProfit.orderId(parent.orderId() + 1);
    takeProfit.action(action.equals("BUY") ? "SELL" : "BUY");
    takeProfit.orderType("LMT");
  ```

```
        takeProfit.totalQuantity(quantity);
        takeProfit.lmtPrice(takeProfitLimitPrice);
        takeProfit.parentId(parentOrderId);
        takeProfit.transmit(false);

        Order stopLoss = new Order();
        stopLoss.orderId(parent.orderId() + 2);
        stopLoss.action(action.equals("BUY") ? "SELL" : "BUY");
        stopLoss.orderType("STP");
        //Stop trigger price
        stopLoss.auxPrice(stopLossPrice);
        stopLoss.totalQuantity(quantity);
        stopLoss.parentId(parentOrderId);
        //In this case, the low side order will be the last child being sent. Therefore, it needs to set
       this attribute to true
        //to activate all its predecessors
        stopLoss.transmit(true);

        List<Order> bracketOrder = new ArrayList<>();
        bracketOrder.add(parent);
        bracketOrder.add(takeProfit);
        bracketOrder.add(stopLoss);

        return bracketOrder;
    }
```

...

```
        List<Order> bracket = OrderSamples.BracketOrder(nextOrderId++, "BUY", 100, 30, 40, 20);
        for(Order o : bracket) {
            client.placeOrder(o.orderId(), ContractSamples.EuropeanStock(), o);
        }
```

- ```
            'This will be our main Or "parent" order
            Dim parent As Order = New Order
            parent.OrderId = parentOrderId
            parent.Action = action
            parent.OrderType = "LMT"
            parent.TotalQuantity = quantity
            parent.LmtPrice = limitPrice
            'The parent And children orders will need this attribute set to false to prevent accidental
       executions.
            'The LAST CHILD will have it set to true,
            parent.Transmit = False

            Dim takeProfit As Order = New Order
            '
            takeProfit.OrderId = parent.OrderId + 1

            If action.Equals("BUY") Then takeProfit.Action = "SELL" Else takeProfit.Action = "BUY"
            takeProfit.OrderType = "LMT"
            takeProfit.TotalQuantity = quantity
            takeProfit.LmtPrice = takeProfitLimitPrice
            takeProfit.ParentId = parentOrderId
            takeProfit.Transmit = False

            Dim stopLoss As Order = New Order
            stopLoss.OrderId = parent.OrderId + 2
            If action.Equals("BUY") Then stopLoss.Action = "SELL" Else stopLoss.Action = "BUY"
            stopLoss.OrderType = "STP"
            'Stop trigger price
            stopLoss.AuxPrice = stopLossPrice
            stopLoss.TotalQuantity = quantity
            stopLoss.ParentId = parentOrderId
            'In this case, the low side order will be the last child being sent. Therefore, it needs to set
       this attribute to true
            'to activate all its predecessors
            stopLoss.Transmit = True

            Dim orders As List(Of Order) = New List(Of Order)
            orders.Add(parent)
            orders.Add(takeProfit)
            orders.Add(stopLoss)
```

...

```
        Dim bracket As List(Of Order) = OrderSamples.BracketOrder(increment(nextOrderId), "BUY", 100, 30,
       40, 20)
        For Each o As Order In bracket
            client.placeOrder(o.OrderId, ContractSamples.EuropeanStock(), o)
        Next
```

- ```
  void OrderSamples::BracketOrder(int parentOrderId, Order& parent, Order& takeProfit, Order& stopLoss,
        std::string action, double quantity, double limitPrice, double takeProfitLimitPrice, double stopLossPrice){
      //This will be our main or "parent" order
  ```

```
        parent.orderId = parentOrderId;
        parent.action = action;
        parent.orderType = "LMT";
        parent.totalQuantity = quantity;
        parent.lmtPrice = limitPrice;
        //The parent and children orders will need this attribute set to false to prevent accidental
           executions.
        //The LAST CHILD will have it set to true,
        parent.transmit = false;

        takeProfit.orderId = parent.orderId + 1;
        takeProfit.action = (action == "BUY") ? "SELL" : "BUY";
        takeProfit.orderType = "LMT";
        takeProfit.totalQuantity = quantity;
        takeProfit.lmtPrice = takeProfitLimitPrice;
        takeProfit.parentId = parentOrderId;
        takeProfit.transmit = false;

        stopLoss.orderId = parent.orderId + 2;
        stopLoss.action = (action == "BUY") ? "SELL" : "BUY";
        stopLoss.orderType = "STP";
        //Stop trigger price
        stopLoss.auxPrice = stopLossPrice;
        stopLoss.totalQuantity = quantity;
        stopLoss.parentId = parentOrderId;
        //In this case, the low side order will be the last child being sent. Therefore, it needs to set this
           attribute to true
        //to activate all its predecessors
        stopLoss.transmit = true;
    }

    ...

        OrderSamples::BracketOrder(m_orderId++, parent, takeProfit, stopLoss, "BUY", 100, 30, 40, 20);
        m_pClient->placeOrder(parent.orderId, ContractSamples::EuropeanStock(), parent);
        m_pClient->placeOrder(takeProfit.orderId, ContractSamples::EuropeanStock(), takeProfit);
        m_pClient->placeOrder(stopLoss.orderId, ContractSamples::EuropeanStock(), stopLoss);
```

-
```
        @staticmethod
        def BracketOrder(parentOrderId:int, action:str, quantity:float,
                         limitPrice:float, takeProfitLimitPrice:float,
                         stopLossPrice:float):

            #This will be our main or "parent" order
            parent = Order()
            parent.orderId = parentOrderId
            parent.action = action
            parent.orderType = "LMT"
            parent.totalQuantity = quantity
            parent.lmtPrice = limitPrice
            #The parent and children orders will need this attribute set to False to prevent accidental
           executions.
            #The LAST CHILD will have it set to True,
            parent.transmit = False

            takeProfit = Order()
            takeProfit.orderId = parent.orderId + 1
            takeProfit.action = "SELL" if action == "BUY" else "BUY"
            takeProfit.orderType = "LMT"
            takeProfit.totalQuantity = quantity
            takeProfit.lmtPrice = takeProfitLimitPrice
            takeProfit.parentId = parentOrderId
            takeProfit.transmit = False

            stopLoss = Order()
            stopLoss.orderId = parent.orderId + 2
            stopLoss.action = "SELL" if action == "BUY" else "BUY"
            stopLoss.orderType = "STP"
            #Stop trigger price
            stopLoss.auxPrice = stopLossPrice
            stopLoss.totalQuantity = quantity
            stopLoss.parentId = parentOrderId
            #In this case, the low side order will be the last child being sent. Therefore, it needs to set
           this attribute to True
            #to activate all its predecessors
            stopLoss.transmit = True

            bracketOrder = [parent, takeProfit, stopLoss]
            return bracketOrder


    ...

            bracket = OrderSamples.BracketOrder(self.nextOrderId(), "BUY", 100, 30, 40, 20)
            for o in bracket:
                self.placeOrder(o.orderId, ContractSamples.EuropeanStock(), o)
                self.nextOrderId()  # need to advance this we'll skip one extra oid, it's fine
```

10.2.5 One Cancels All

The `One-Cancels All` (OCA) order type allows an investor to place multiple and possibly unrelated orders assigned to a group. The aim is to complete just one of the orders, which in turn will cause TWS to cancel the remaining orders. The investor may submit several orders aimed at taking advantage of the most desirable price within the group. Completion of one piece of the group order causes cancellation of the remaining group orders while partial completion causes the group to re-balance. An investor might desire to sell 1000 shares of only ONE of three positions held above prevailing market prices. The OCA order group allows the investor to enter prices at specified target levels and if one is completed, the other two will automatically cancel. Alternatively, an investor may wish to take a LONG position in eMini S&P stock index futures in a falling market or else SELL US treasury futures at a more favorable price. Grouping the two orders using an OCA order type offers the investor two chances to enter a similar position, while only running the risk of taking on a single position.

-
```
        public static List<Order> OneCancelsAll(string ocaGroup, List<Order> ocaOrders, int ocaType)
        {
            foreach (Order o in ocaOrders)
            {
                o.OcaGroup = ocaGroup;
                o.OcaType = ocaType;
            }
            return ocaOrders;
        }
```

...

```
            List<Order> ocaOrders = new List<Order>();
            ocaOrders.Add(OrderSamples.LimitOrder("BUY", 1, 10));
            ocaOrders.Add(OrderSamples.LimitOrder("BUY", 1, 11));
            ocaOrders.Add(OrderSamples.LimitOrder("BUY", 1, 12));
            OrderSamples.OneCancelsAll("TestOCA_" + nextOrderId, ocaOrders, 2);
            foreach (Order o in ocaOrders)
                client.placeOrder(nextOrderId++, ContractSamples.USStock(), o);
```

-
```
    public static List<Order> OneCancelsAll(String ocaGroup, List<Order> ocaOrders, int ocaType) {

        for (Order o : ocaOrders) {
            o.ocaGroup(ocaGroup);
            o.ocaType(ocaType);
        }
        return ocaOrders;
    }
```

...

```
        List<Order> OcaOrders = new ArrayList<>();
        OcaOrders.add(OrderSamples.LimitOrder("BUY", 1, 10));
        OcaOrders.add(OrderSamples.LimitOrder("BUY", 1, 11));
        OcaOrders.add(OrderSamples.LimitOrder("BUY", 1, 12));
        OcaOrders = OrderSamples.OneCancelsAll("TestOCA_" + nextOrderId, OcaOrders, 2);
        for (Order o : OcaOrders) {

            client.placeOrder(nextOrderId++, ContractSamples.USStock(), o);
        }
```

-
```
            For Each o As Order In ocaOrders

                o.OcaGroup = ocaGroup
                o.OcaType = ocaType
                'Same as with Bracket orders. To prevent accidental executions, set all orders' transmit
        flag to false.
                'This will tell the TWS Not to send the orders, allowing your program to send them all
        first.
                o.Transmit = False
            Next o

            'Telling the TWS to transmit the last order in the OCA will also cause the transmission of its
        predecessors.
            ocaOrders.Item(ocaOrders.Count - 1).Transmit = True
```

...

```
        Dim ocaOrders As List(Of Order) = New List(Of Order)
        ocaOrders.Add(OrderSamples.LimitOrder("BUY", 1, 10))
        ocaOrders.Add(OrderSamples.LimitOrder("BUY", 1, 11))
        ocaOrders.Add(OrderSamples.LimitOrder("BUY", 1, 12))
        OrderSamples.OneCancelsAll("TestOCA_" + nextOrderId, ocaOrders, 2)
        For Each o As Order In ocaOrders
            client.placeOrder(increment(nextOrderId), ContractSamples.USStock(), o)
        Next
```

- ```
  void OrderSamples::OneCancelsAll(std::string ocaGroup, Order& ocaOrder, int ocaType){
          ocaOrder.ocaGroup = ocaGroup;
          ocaOrder.ocaType = ocaType;
  }
  ```

  ...

  ```
      std::vector<Order> ocaOrders;
      ocaOrders.push_back(OrderSamples::LimitOrder("BUY", 1, 10));
      ocaOrders.push_back(OrderSamples::LimitOrder("BUY", 1, 11));
      ocaOrders.push_back(OrderSamples::LimitOrder("BUY", 1, 12));
      for(unsigned int i = 0; i < ocaOrders.size(); i++){
          OrderSamples::OneCancelsAll("TestOca", ocaOrders[i], 2);
          m_pClient->placeOrder(m_orderId++, ContractSamples::USStock(), ocaOrders[i]);
      }
  ```

- ```
      @staticmethod
      def OneCancelsAll(ocaGroup:str, ocaOrders:ListOfOrder, ocaType:int):

          for o in ocaOrders:

              o.ocaGroup = ocaGroup
              o.ocaType = ocaType

          return ocaOrders
  ```

  ...

  ```
          ocaOrders = [OrderSamples.LimitOrder("BUY", 1, 10), OrderSamples.LimitOrder("BUY", 1, 11),
                       OrderSamples.LimitOrder("BUY", 1, 12)]
          OrderSamples.OneCancelsAll("TestOCA_" + self.nextValidOrderId, ocaOrders, 2)
          for o in ocaOrders:
              self.placeOrder(self.nextOrderId(), ContractSamples.USStock(), o)
  ```

10.2.5.0.1 OCA Types

Via the **IBApi.Order.OcaType** attribute, the way in which remaining orders should be handled after an execution can be configured as indicated in the table below:

Value	Description
1	Cancel all remaining orders with block.∗
2	Remaining orders are proportionately reduced in size with block.∗
3	Remaining orders are proportionately reduced in size with no block.

Note∗: if you use a value "with block" gives your order has overfill protection. This means that only one order in the group will be routed at a time to remove the possibility of an overfill.

10.2.6 Adjustable Stops

You can attach one-time adjustments to stop, stop limit, trailing stop and trailing stop limit orders. When you attach an adjusted order, you set a trigger price that triggers a modification of the original (or parent) order, instead of triggering order transmission.

10.2.6.0.1 Adjusted to Stop

- ```
        //Attached order is a conventional STP order in opposite direction
        Order order = Stop(parent.Action.Equals("BUY") ? "SELL" : "BUY", parent.TotalQuantity,
  attachedOrderStopPrice);
        order.ParentId = parent.OrderId;
        //When trigger price is penetrated
        order.TriggerPrice = triggerPrice;
        //The parent order will be turned into a STP order
        order.AdjustedOrderType = "STP";
        //With the given STP price
        order.AdjustedStopPrice = adjustStopPrice;
  ```

- ```
      Order order = new Order();
      //Attached order is a conventional STP order in opposite direction
      order.action("BUY".equals(parent.getAction()) ? "SELL" : "BUY");
      order.totalQuantity(parent.totalQuantity());
      order.auxPrice(attachedOrderStopPrice);
      order.parentId(parent.orderId());
      //When trigger price is penetrated
      order.triggerPrice(triggerPrice);
      //The parent order will be turned into a STP order
      order.adjustedOrderType(OrderType.STP);
      //With the given STP price
      order.adjustedStopPrice(adjustStopPrice);
  ```

- ```
        'Attached order Is a conventional STP order in opposite direction
        Dim action As String = "BUY"
        If (parent.Action.Equals("BUY")) Then
            action = "SELL"
        End If
        Dim order As Order = StopOrder(action, parent.TotalQuantity, attachedOrderStopPrice)
        order.ParentId = parent.OrderId
        'When trigger price Is penetrated
        order.TriggerPrice = triggerPrice
        'The parent order will be turned into a STP order
        order.AdjustedOrderType = "STP"
        'With the given STP price
        order.AdjustedStopPrice = adjustStopPrice
  ```

- ```
  //Attached order is a conventional STP order in opposite direction
  Order order;
  order.action = (parent.action == "BUY") ? "SELL": "BUY";
  order.orderType = "STP";
  order.totalQuantity = parent.totalQuantity;
  order.auxPrice = attachedOrderStopPrice;
  order.parentId = parent.orderId;
  //When trigger price is penetrated
  order.triggerPrice = triggerPrice;
  //The parent order will be turned into a STP order
  order.adjustedOrderType = "STP";
  //With the given STP price
  order.adjustedStopPrice = adjustStopPrice;
  ```

- ```
      # Attached order is a conventional STP order in opposite direction
      order = OrderSamples.Stop("SELL" if parent.action == "BUY" else "BUY",
                  parent.totalQuantity, attachedOrderStopPrice)
      order.parentId = parent.orderId
      #When trigger price is penetrated
      order.triggerPrice = triggerPrice
      #The parent order will be turned into a STP order
      order.adjustedOrderType = "STP"
      #With the given STP price
      order.adjustedStopPrice = adjustStopPrice
  ```

10.2.6.0.2 Adjusted to Stop Limit

- ```
        //Attached order is a conventional STP order
        Order order = Stop(parent.Action.Equals("BUY") ? "SELL" : "BUY", parent.TotalQuantity,
  attachedOrderStopPrice);
        order.ParentId = parent.OrderId;
        //When trigger price is penetrated
        order.TriggerPrice = triggerPrice;
        //The parent order will be turned into a STP LMT order
        order.AdjustedOrderType = "STP LMT";
        //With the given stop price
        order.AdjustedStopPrice = adjustedStopPrice;
        //And the given limit price
        order.AdjustedStopLimitPrice = adjustedStopLimitPrice;
  ```

- ```
        Order order = new Order();
        //Attached order is a conventional STP order
        order.action("BUY".equals(parent.getAction()) ? "SELL" : "BUY");
        order.totalQuantity(parent.totalQuantity());
        order.auxPrice(attachedOrderStopPrice);
        order.parentId(parent.orderId());
        //When trigger price is penetrated
        order.triggerPrice(triggerPrice);
        //The parent order will be turned into a STP LMT order
        order.adjustedOrderType(OrderType.STP_LMT);
        //With the given stop price
        order.adjustedStopPrice(adjustStopPrice);
        //And the given limit price
        order.adjustedStopLimitPrice(adjustedStopLimitPrice);
  ```

- ```
            'Attached order Is a conventional STP order
            Dim action As String = "BUY"
            If (parent.Action.Equals("BUY")) Then
                action = "SELL"
            End If
            Dim order As Order = StopOrder(action, parent.TotalQuantity, attachedOrderStopPrice)
            order.ParentId = parent.OrderId
            'When trigger price Is penetrated
            order.TriggerPrice = triggerPrice
            'The parent order will be turned into a STP LMT order
            order.AdjustedOrderType = "STP LMT"
            'With the given stop price
            order.AdjustedStopPrice = adjustedStopPrice
            'And the given limit price
            order.AdjustedStopLimitPrice = adjustedStopLimitPrice
  ```

- ```
    //Attached order is a conventional STP order
    Order order;
    order.action = (parent.action == "BUY") ? "SELL": "BUY";
    order.orderType = "STP";
    order.totalQuantity = parent.totalQuantity;
    order.auxPrice = attachedOrderStopPrice;
    order.parentId = parent.orderId;
    //When trigger price is penetrated
    order.triggerPrice = triggerPrice;
    //The parent order will be turned into a STP LMT order
    order.adjustedOrderType = "STP LMT";
    //With the given stop price
    order.adjustedStopPrice = adjustStopPrice;
    //And the given limit price
    order.adjustedStopLimitPrice = adjustedStopLimitPrice;
  ```

- ```
        #Attached order is a conventional STP order
        order = OrderSamples.Stop("SELL" if parent.action == "BUY" else "BUY",
                                  parent.totalQuantity, attachedOrderStopPrice)
        order.parentId = parent.orderId
        #When trigger price is penetrated
        order.triggerPrice = triggerPrice
        #The parent order will be turned into a STP LMT order
        order.adjustedOrderType = "STP LMT"
        #With the given stop price
        order.adjustedStopPrice = adjustedStopPrice
        #And the given limit price
        order.adjustedStopLimitPrice = adjustedStopLimitPrice
  ```

10.2.6.0.3 Adjusted to Trail

- ```
            //Attached order is a conventional STP order
            Order order = Stop(parent.Action.Equals("BUY") ? "SELL" : "BUY", parent.TotalQuantity,
      attachedOrderStopPrice);
            order.ParentId = parent.OrderId;
            //When trigger price is penetrated
            order.TriggerPrice = triggerPrice;
            //The parent order will be turned into a TRAIL order
            order.AdjustedOrderType = "TRAIL";
            //With a stop price of...
            order.AdjustedStopPrice = adjustedStopPrice;
            //traling by and amount (0) or a percent (1)...
            order.AdjustableTrailingUnit = trailUnit;
            //of...
            order.AdjustedTrailingAmount = adjustedTrailAmount;
  ```

- ```
        Order order = new Order();
        //Attached order is a conventional STP order
        order.action("BUY".equals(parent.getAction()) ? "SELL" : "BUY");
        order.totalQuantity(parent.totalQuantity());
        order.auxPrice(attachedOrderStopPrice);
        order.parentId(parent.orderId());
        //When trigger price is penetrated
  ```

```
            order.triggerPrice(triggerPrice);
            //The parent order will be turned into a TRAIL order
            order.adjustedOrderType(OrderType.TRAIL);
            //With a stop price of...
            order.adjustedStopPrice(adjustStopPrice);
            //trailing by and amount (0) or a percent (1)...
            order.adjustableTrailingUnit(trailUnit);
            //of...
            order.adjustedTrailingAmount(adjustedTrailAmount);
```

```
            'Attached order Is a conventional STP order
            Dim action As String = "BUY"
            If (parent.Action.Equals("BUY")) Then
                action = "SELL"
            End If
            Dim order As Order = StopOrder(action, parent.TotalQuantity, attachedOrderStopPrice)
            order.ParentId = parent.OrderId
            'When trigger price Is penetrated
            order.TriggerPrice = triggerPrice
            'The parent order will be turned into a TRAIL order
            order.AdjustedOrderType = "TRAIL"
            'With a stop price of...
            order.AdjustedStopPrice = adjustedStopPrice
            'traling by And amount (0) Or a percent (1)...
            order.AdjustableTrailingUnit = trailUnit
            'of...
            order.AdjustedTrailingAmount = adjustedTrailAmount
```

```
    //Attached order is a conventional STP order
    Order order;
    order.action = (parent.action == "BUY") ? "SELL": "BUY";
    order.orderType = "STP";
    order.totalQuantity = parent.totalQuantity;
    order.auxPrice = attachedOrderStopPrice;
    order.parentId = parent.orderId;
    //When trigger price is penetrated
    order.triggerPrice = triggerPrice;
    //The parent order will be turned into a TRAIL order
    order.adjustedOrderType = "TRAIL";
    //With a stop price of...
    order.adjustedStopPrice = adjustStopPrice;
    //traling by and amount (0) or a percent (1)...
    order.adjustableTrailingUnit = trailUnit;
    //of...
    order.adjustedTrailingAmount = adjustedTrailAmount;
```

```
        #Attached order is a conventional STP order
        order = OrderSamples.Stop("SELL" if parent.action == "BUY" else "BUY",
                    parent.totalQuantity, attachedOrderStopPrice)
        order.parentId = parent.orderId
        #When trigger price is penetrated
        order.triggerPrice = triggerPrice
        #The parent order will be turned into a TRAIL order
        order.adjustedOrderType = "TRAIL"
        #With a stop price of...
        order.adjustedStopPrice = adjustedStopPrice
        #traling by and amount (0) or a percent (1)...
        order.adjustableTrailingUnit = trailUnit
        #of...
        order.adjustedTrailingAmount = adjustedTrailAmount
```

10.2.7 Order Conditioning

Conditions allow to activate orders given a certain criteria...

```
            Order mkt = OrderSamples.MarketOrder("BUY", 100);
            //Order will become active if conditioning criteria is met
            mkt.ConditionsCancelOrder = true;
            mkt.Conditions.Add(OrderSamples.PriceCondition(208813720, "SMART", 600, false, false));
            mkt.Conditions.Add(OrderSamples.ExecutionCondition("EUR.USD", "CASH", "IDEALPRO", true));
            mkt.Conditions.Add(OrderSamples.MarginCondition(30, true, false));
            mkt.Conditions.Add(OrderSamples.PercentageChangeCondition(15.0, 208813720, "SMART", true, true)
    );
            mkt.Conditions.Add(OrderSamples.TimeCondition("20160118 23:59:59", true, false));
            mkt.Conditions.Add(OrderSamples.VolumeCondition(208813720, "SMART", false, 100, true));
            client.placeOrder(nextOrderId++, ContractSamples.EuropeanStock(), mkt);
```

- ```
        Order mkt = OrderSamples.MarketOrder("BUY", 100);
        //Order will become active if conditioning criteria is met
        mkt.conditionsCancelOrder(true);
        mkt.conditions().add(OrderSamples.PriceCondition(208813720, "SMART", 600, false, false));
        mkt.conditions().add(OrderSamples.ExecutionCondition("EUR.USD", "CASH", "IDEALPRO", true));
        mkt.conditions().add(OrderSamples.MarginCondition(30, true, false));
        mkt.conditions().add(OrderSamples.PercentageChangeCondition(15.0, 208813720, "SMART", true, true));
        mkt.conditions().add(OrderSamples.TimeCondition("20160118 23:59:59", true, false));
        mkt.conditions().add(OrderSamples.VolumeCondition(208813720, "SMART", false, 100, true));
        client.placeOrder(nextOrderId++, ContractSamples.EuropeanStock(), mkt);
  ```

- ```
        Dim mkt As Order = OrderSamples.MarketOrder("BUY", 100)
        'Order will become active if conditioning criteria Is met
        mkt.ConditionsCancelOrder = True
        mkt.Conditions.Add(OrderSamples.PriceCondition(208813720, "SMART", 600, False, False))
        mkt.Conditions.Add(OrderSamples.ExecutionCondition("EUR.USD", "CASH", "IDEALPRO", True))
        mkt.Conditions.Add(OrderSamples.MarginCondition(30, True, False))
        mkt.Conditions.Add(OrderSamples.PercentageChangeCondition(15.0, 208813720, "SMART", True, True))
        mkt.Conditions.Add(OrderSamples.TimeCondition("20160118 23:59:59", True, False))
        mkt.Conditions.Add(OrderSamples.VolumeCondition(208813720, "SMART", False, 100, True))
        client.placeOrder(increment(nextOrderId), ContractSamples.EuropeanStock(), mkt)
  ```

- ```
    Order lmt = OrderSamples::LimitOrder("BUY", 100, 10);
    //Order will become active if conditioning criteria is met
    PriceCondition* priceCondition = dynamic_cast<PriceCondition *>(OrderSamples::Price_Condition(208813720
      , "SMART", 600, false, false));
    ExecutionCondition* execCondition = dynamic_cast<ExecutionCondition *>(
      OrderSamples::Execution_Condition("EUR.USD", "CASH", "IDEALPRO", true));
    MarginCondition* marginCondition = dynamic_cast<MarginCondition *>(OrderSamples::Margin_Condition(30,
      true, false));
    PercentChangeCondition* pctChangeCondition = dynamic_cast<PercentChangeCondition *>(
      OrderSamples::Percent_Change_Condition(15.0, 208813720, "SMART", true, true));
    TimeCondition* timeCondition = dynamic_cast<TimeCondition *>(OrderSamples::Time_Condition("20160118
       23:59:59", true, false));
    VolumeCondition* volumeCondition = dynamic_cast<VolumeCondition *>(OrderSamples::Volume_Condition(20881
      3720, "SMART", false, 100, true));

    lmt.conditions.push_back(std::shared_ptr<PriceCondition>(priceCondition));
    lmt.conditions.push_back(std::shared_ptr<ExecutionCondition>(execCondition));
    lmt.conditions.push_back(std::shared_ptr<MarginCondition>(marginCondition));
    lmt.conditions.push_back(std::shared_ptr<PercentChangeCondition>(pctChangeCondition));
    lmt.conditions.push_back(std::shared_ptr<TimeCondition>(timeCondition));
    lmt.conditions.push_back(std::shared_ptr<VolumeCondition>(volumeCondition));
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStock(),lmt);
  ```

- ```
        mkt = OrderSamples.MarketOrder("BUY", 100)
        # Order will become active if conditioning criteria is met
        mkt.conditionsCancelOrder = True
        mkt.conditions.append(
            OrderSamples.PriceCondition(PriceCondition.TriggerMethodEnum.Default,
                                        208813720, "SMART", 600, False, False))
        mkt.conditions.append(OrderSamples.ExecutionCondition("EUR.USD", "CASH", "IDEALPRO", True))
        mkt.conditions.append(OrderSamples.MarginCondition(30, True, False))
        mkt.conditions.append(OrderSamples.PercentageChangeCondition(15.0, 208813720, "SMART", True, True))
        mkt.conditions.append(OrderSamples.TimeCondition("20160118 23:59:59", True, False))
        mkt.conditions.append(OrderSamples.VolumeCondition(208813720, "SMART", False, 100, True))
        self.placeOrder(self.nextOrderId(), ContractSamples.EuropeanStock(), mkt)
  ```

Or cancel them

- ```
            Order lmt = OrderSamples.LimitOrder("BUY", 100, 20);
            //The active order will be cancelled if conditioning criteria is met
            lmt.ConditionsCancelOrder = true;
            lmt.Conditions.Add(OrderSamples.PriceCondition(208813720, "SMART", 600, false, false));
            client.placeOrder(nextOrderId++, ContractSamples.EuropeanStock(), lmt);
  ```

- ```
        Order lmt = OrderSamples.LimitOrder("BUY", 100, 20);
        //The active order will be cancelled if conditioning criteria is met
        lmt.conditionsCancelOrder(true);
        lmt.conditions().add(OrderSamples.PriceCondition(208813720, "SMART", 600, false, false));
        client.placeOrder(nextOrderId++, ContractSamples.EuropeanStock(), lmt);
  ```

- ```
        Dim lmt As Order = OrderSamples.LimitOrder("BUY", 100, 20)
        'The active order will be cancelled if conditioning criteria Is met
        lmt.ConditionsCancelOrder = True
        lmt.Conditions.Add(OrderSamples.PriceCondition(208813720, "SMART", 600, False, False))
        client.placeOrder(increment(nextOrderId), ContractSamples.EuropeanStock(), lmt)
  ```

- ```
    Order lmt2 = OrderSamples::LimitOrder("BUY", 100, 20);
    //The active order will be cancelled if conditioning criteria is met
    lmt2.conditionsCancelOrder = true;
    PriceCondition* priceCondition2 = dynamic_cast<PriceCondition *>(OrderSamples::Price_Condition(20881372
      0, "SMART", 600, false, false));
    lmt2.conditions.push_back(std::shared_ptr<PriceCondition>(priceCondition2));
    m_pClient->placeOrder(m_orderId++, ContractSamples::EuropeanStock(), lmt2);
  ```

-
```
        lmt = OrderSamples.LimitOrder("BUY", 100, 20)
        # The active order will be cancelled if conditioning criteria is met
        lmt.conditionsCancelOrder = True
        lmt.conditions.append(
            OrderSamples.PriceCondition(PriceCondition.TriggerMethodEnum.Last,
                                        208813720, "SMART", 600, False, False))
        self.placeOrder(self.nextOrderId(), ContractSamples.EuropeanStock(), lmt)
```

10.2.7.0.1 Price Conditions

-
```
            //Conditions have to be created via the OrderCondition.Create
            PriceCondition priceCondition = (PriceCondition)OrderCondition.Create(OrderConditionType.Price)
        ;
            //When this contract...
            priceCondition.ConId = conId;
            //traded on this exchange
            priceCondition.Exchange = exchange;
            //has a price above/below
            priceCondition.IsMore = isMore;
            //this quantity
            priceCondition.Price = price;
            //AND | OR next condition (will be ignored if no more conditions are added)
            priceCondition.IsConjunctionConnection = isConjunction;
```

-
```
        //Conditions have to be created via the OrderCondition.Create
        PriceCondition priceCondition = (PriceCondition)OrderCondition.create(OrderConditionType.Price);
        //When this contract...
        priceCondition.conId(conId);
        //traded on this exchange
        priceCondition.exchange(exchange);
        //has a price above/below
        priceCondition.isMore(isMore);
        //this quantity
        priceCondition.price(price);
        //AND | OR next condition (will be ignored if no more conditions are added)
        priceCondition.conjunctionConnection(isConjunction);
```

-
```
            'Conditions have to be created via the OrderCondition.Create
            Dim _priceCondition As PriceCondition = OrderCondition.Create(OrderConditionType.Price) 'cast
       to priceCondition
            'When this contract...
            _priceCondition.ConId = conId
            'traded on this exchange
            _priceCondition.Exchange = exchange
            'has a price above/below
            _priceCondition.IsMore = isMore
            'this quantity
            _priceCondition.Price = price
            'And | Or next condition (will be ignored if no more conditions are added)
            _priceCondition.IsConjunctionConnection = isConjunction
```

-
```
    //Conditions have to be created via the OrderCondition.Create
    PriceCondition* priceCondition = dynamic_cast<PriceCondition *>(OrderCondition::create(
      OrderCondition::OrderConditionType::Price));
    //When this contract...
    priceCondition->conId(conId);
    //traded on this exchange
    priceCondition->exchange(exchange);
    //has a price above/below
    priceCondition->isMore(isMore);
    //this quantity
    priceCondition->price(price);
    //AND | OR next condition (will be ignored if no more conditions are added)
    priceCondition->conjunctionConnection(isConjunction);
```

-
```
        #Conditions have to be created via the OrderCondition.create
        priceCondition = order_condition.Create(OrderCondition.Price)
        #When this contract...
        priceCondition.conId = conId
        #traded on this exchange
        priceCondition.exchange = exchange
        #has a price above/below
        priceCondition.isMore = isMore
        priceCondition.triggerMethod = triggerMethod
        #this quantity
        priceCondition.price = price
        #AND | OR next condition (will be ignored if no more conditions are added)
        priceCondition.isConjunctionConnection = isConjunction
```

10.2.7.0.2 Execution Conditions

- ```
            ExecutionCondition execCondition = (ExecutionCondition)OrderCondition.Create(OrderConditionType
      .Execution);
            //When an execution on symbol
            execCondition.Symbol = symbol;
            //at exchange
            execCondition.Exchange = exchange;
            //for this secType
            execCondition.SecType = secType;
            //AND | OR next condition (will be ignored if no more conditions are added)
            execCondition.IsConjunctionConnection = isConjunction;
  ```

- ```
        ExecutionCondition execCondition = (ExecutionCondition)OrderCondition.create(OrderConditionType.
      Execution);
        //When an execution on symbol
        execCondition.symbol(symbol);
        //at exchange
        execCondition.exchange(exchange);
        //for this secType
        execCondition.secType(secType);
        //AND | OR next condition (will be ignored if no more conditions are added)
        execCondition.conjunctionConnection(isConjunction);
  ```

- ```
            Dim execCondition As ExecutionCondition = OrderCondition.Create(OrderConditionType.Execution) '
       cast to (ExecutionCondition)
            'When an execution on symbol
            execCondition.Symbol = symbol
            'at exchange
            execCondition.Exchange = exchange
            'for this secType
            execCondition.SecType = secType
            'And | Or next condition (will be ignored if no more conditions are added)
            execCondition.IsConjunctionConnection = isConjunction
  ```

- ```
  ExecutionCondition* execCondition = dynamic_cast<ExecutionCondition *>(OrderCondition::create(
    OrderCondition::OrderConditionType::Execution));
  //When an execution on symbol
  execCondition->symbol(symbol);
  //at exchange
  execCondition->exchange(exchange);
  //for this secType
  execCondition->secType(secType);
  //AND | OR next condition (will be ignored if no more conditions are added)
  execCondition->conjunctionConnection(isConjunction);
  ```

- ```
      execCondition = order_condition.Create(OrderCondition.Execution)
      #When an execution on symbol
      execCondition.symbol = symbol
      #at exchange
      execCondition.exchange = exchange
      #for this secType
      execCondition.secType = secType
      #AND | OR next condition (will be ignored if no more conditions are added)
      execCondition.isConjunctionConnection = isConjunction
  ```

10.2.7.0.3 Margin Conditions

- ```
            MarginCondition marginCondition = (MarginCondition)OrderCondition.Create(OrderConditionType.
      Margin);
            //If margin is above/below
            marginCondition.IsMore = isMore;
            //given percent
            marginCondition.Percent = percent;
            //AND | OR next condition (will be ignored if no more conditions are added)
            marginCondition.IsConjunctionConnection = isConjunction;
  ```

- ```
        MarginCondition marginCondition = (MarginCondition)OrderCondition.create(OrderConditionType.Margin)
      ;
        //If margin is above/below
        marginCondition.isMore(isMore);
        //given percent
        marginCondition.percent(percent);
        //AND | OR next condition (will be ignored if no more conditions are added)
        marginCondition.conjunctionConnection(isConjunction);
  ```

- ```
            Dim _MarginCondition As MarginCondition = OrderCondition.Create(OrderConditionType.Margin) '
       cast to (MarginCondition)
            'If margin Is above/below
            _MarginCondition.IsMore = isMore
            'given percent
            _MarginCondition.Percent = percent
            'And | Or next condition (will be ignored if no more conditions are added)
            _MarginCondition.IsConjunctionConnection = isConjunction
  ```

- ```
  MarginCondition* marginCondition = dynamic_cast<MarginCondition *>(OrderCondition::create(
    OrderCondition::OrderConditionType::Margin));
  //If margin is above/below
  marginCondition->percent(percent);
  //given percent
  marginCondition->isMore(isMore);
  //AND | OR next condition (will be ignored if no more conditions are added)
  marginCondition->conjunctionConnection(isConjunction);
  ```

- ```
      marginCondition = order_condition.Create(OrderCondition.Margin)
      #If margin is above/below
      marginCondition.isMore = isMore
      #given percent
      marginCondition.percent = percent
      #AND | OR next condition (will be ignored if no more conditions are added)
      marginCondition.isConjunctionConnection = isConjunction
  ```

10.2.7.0.4 Percentage Conditions

- ```
          PercentChangeCondition pctChangeCondition = (PercentChangeCondition)OrderCondition.Create(
    OrderConditionType.PercentCange);
          //If there is a price percent change measured against last close price above or below...
          pctChangeCondition.IsMore = isMore;
          //this amount...
          pctChangeCondition.ChangePercent = pctChange;
          //on this contract
          pctChangeCondition.ConId = conId;
          //when traded on this exchange...
          pctChangeCondition.Exchange = exchange;
          //AND | OR next condition (will be ignored if no more conditions are added)
          pctChangeCondition.IsConjunctionConnection = isConjunction;
  ```

- ```
      PercentChangeCondition pctChangeCondition = (PercentChangeCondition)OrderCondition.create(
    OrderConditionType.PercentChange);
      //If there is a price percent change measured against last close price above or below...
      pctChangeCondition.isMore(isMore);
      //this amount...
      pctChangeCondition.changePercent(pctChange);
      //on this contract
      pctChangeCondition.conId(conId);
      //when traded on this exchange...
      pctChangeCondition.exchange(exchange);
      //AND | OR next condition (will be ignored if no more conditions are added)
      pctChangeCondition.conjunctionConnection(isConjunction);
  ```

- ```
          Dim pctChangeCondition As PercentChangeCondition =
     OrderCondition.Create(OrderConditionType.PercentCange) 'cast (PercentChangeCondition)
          'If there Is a price percent change measured against last close price above Or below...
          pctChangeCondition.IsMore = isMore
          'this amount...
          pctChangeCondition.ChangePercent = pctChange
          'on this contract
          pctChangeCondition.ConId = conId
          'when traded on this exchange...
          pctChangeCondition.Exchange = exchange
          'And | Or next condition (will be ignored if no more conditions are added)
          pctChangeCondition.IsConjunctionConnection = isConjunction
  ```

- ```
  PercentChangeCondition* pctChangeCondition = dynamic_cast<PercentChangeCondition *>(
    OrderCondition::create(OrderCondition::OrderConditionType::PercentChange));
  //If there is a price percent change measured against last close price above or below...
  pctChangeCondition->isMore(isMore);
  //this amount...
  pctChangeCondition->changePercent(pctChange);
  //on this contract
  pctChangeCondition->conId(conId);
  //when traded on this exchange...
  pctChangeCondition->exchange(exchange);
  //AND | OR next condition (will be ignored if no more conditions are added)
  pctChangeCondition->conjunctionConnection(isConjunction);
  ```

- ```
      pctChangeCondition = order_condition.Create(OrderCondition.PercentChange)
      #If there is a price percent change measured against last close price above or below...
      pctChangeCondition.isMore = isMore
      #this amount...
      pctChangeCondition.changePercent = pctChange
      #on this contract
      pctChangeCondition.conId = conId
      #when traded on this exchange...
      pctChangeCondition.exchange = exchange
      #AND | OR next condition (will be ignored if no more conditions are added)
      pctChangeCondition.isConjunctionConnection = isConjunction
  ```
  ```

10.2.7.0.5 Time Conditions

- ```
  TimeCondition timeCondition = (TimeCondition)OrderCondition.Create(OrderConditionType.Time);
  //Before or after...
  timeCondition.IsMore = isMore;
  //this time..
  timeCondition.Time = time;
  //AND | OR next condition (will be ignored if no more conditions are added)
  timeCondition.IsConjunctionConnection = isConjunction;
  ```

- ```
  TimeCondition timeCondition = (TimeCondition)OrderCondition.create(OrderConditionType.Time);
  //Before or after...
  timeCondition.isMore(isMore);
  //this time...
  timeCondition.time(time);
  //AND | OR next condition (will be ignored if no more conditions are added)
  timeCondition.conjunctionConnection(isConjunction);
  ```

- ```
  Dim _TimeCondition As TimeCondition = OrderCondition.Create(OrderConditionType.Time) 'cast
  (timeCondition)
  'Before Or after...
  _TimeCondition.IsMore = isMore
  'this time..
  _TimeCondition.Time = time
  'And | Or next condition (will be ignored if no more conditions are added)
  _TimeCondition.IsConjunctionConnection = isConjunction
  ```

- ```
  TimeCondition* timeCondition = dynamic_cast<TimeCondition *>(OrderCondition::create(
    OrderCondition::OrderConditionType::Time));
  //Before or after...
  timeCondition->isMore(isMore);
  //this time..
  timeCondition->time(time);
  //AND | OR next condition (will be ignored if no more conditions are added)
  timeCondition->conjunctionConnection(isConjunction);
  ```

- ```
  timeCondition = order_condition.Create(OrderCondition.Time)
  #Before or after...
  timeCondition.isMore = isMore
  #this time..
  timeCondition.time = time
  #AND | OR next condition (will be ignored if no more conditions are added)
  timeCondition.isConjunctionConnection = isConjunction
  ```

10.2.7.0.6 Volume Conditions

- ```
  VolumeCondition volCond = (VolumeCondition)OrderCondition.Create(OrderConditionType.Volume);
  //Whenever contract...
  volCond.ConId = conId;
  //When traded at
  volCond.Exchange = exchange;
  //reaches a volume higher/lower
  volCond.IsMore = isMore;
  //than this...
  volCond.Volume = volume;
  //AND | OR next condition (will be ignored if no more conditions are added)
  volCond.IsConjunctionConnection = isConjunction;
  ```

- ```
  VolumeCondition volCon = (VolumeCondition)OrderCondition.create(OrderConditionType.Volume);
  //Whenever contract...
  volCon.conId(conId);
  //When traded at
  volCon.exchange(exchange);
  //reaches a volume higher/lower
  volCon.isMore(isMore);
  //than this...
  volCon.volume(volume);
  //AND | OR next condition (will be ignored if no more conditions are added)
  volCon.conjunctionConnection(isConjunction);
  ```

- ```
  Dim volCond As VolumeCondition = OrderCondition.Create(OrderConditionType.Volume) 'cast
  (VolumeCondition)
  'Whenever contract...
  volCond.ConId = conId
  'When traded at
  volCond.Exchange = exchange
  'reaches a volume higher/lower
  volCond.IsMore = isMore
  'than this...
  volCond.Volume = volume
  'And | Or next condition (will be ignored if no more conditions are added)
  volCond.IsConjunctionConnection = isConjunction
  ```

- ```
  VolumeCondition* volCondition = dynamic_cast<VolumeCondition *>(OrderCondition::create(
    OrderCondition::OrderConditionType::Volume));
  //Whenever contract...
  volCondition->conId(conId);
  //When traded at
  volCondition->exchange(exchange);
  //reaches a volume higher/lower
  volCondition->isMore(isMore);
  //than this...
  volCondition->volume(volume);
  //AND | OR next condition (will be ignored if no more conditions are added)
  volCondition->conjunctionConnection(isConjunction);
  ```

- ```
  volCond = order_condition.Create(OrderCondition.Volume)
  #Whenever contract...
  volCond.conId = conId
  #When traded at
  volCond.exchange = exchange
  #reaches a volume higher/lower
  volCond.isMore = isMore
  #than this...
  volCond.volume = volume
  #AND | OR next condition (will be ignored if no more conditions are added)
  volCond.isConjunctionConnection = isConjunction
  ```

10.2.8 Algorithms

Order behaviour can be further enhanced through Interactive Broker's powerful algorithmic strategies. To make use of our variety of algorithms make use of **IBApi.Order.AlgoStrategy** and **IBApi.Order.AlgoParams** attributes.

- **IB Algorithms**
- **CSFB Algorithms**
- **Jefferies Algorithms**

10.2.9 IB Algorithms

10.2.9.0.0.1 Adaptive Algo

The `Adaptive Algo` combines IB's Smartrouting capabilities with user-defined priority settings in an effort to achieve further cost efficiency at the point of execution. Using the Adaptive algo leads to better execution prices on average than for regular limit or market orders.

Parameter	Description	Values
adaptivePriority	The 'Priority' selector determines the time taken to scan for better execution prices. The 'Urgent' setting scans only briefly, while the 'Patient' scan works more slowly and has a higher chance of achieving a better overall fill for your order.	Urgent > Normal > Patient

- ```
  Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
  ```
  ```

```
...

            AvailableAlgoParams.FillAdaptiveParams(baseOrder, "Normal");
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);


...

        public static void FillAdaptiveParams(Order baseOrder, string priority)
        {
            baseOrder.AlgoStrategy = "Adaptive";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("adaptivePriority", priority));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

```
...

        AvailableAlgoParams.FillAdaptiveParams(baseOrder, "Normal");
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);


...

    public static void FillAdaptiveParams(Order baseOrder, String priority) {

        baseOrder.algoStrategy("Adaptive");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("adaptivePriority", priority));

    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

```
...

        AvailableAlgoParams.FillAdaptiveParams(baseOrder, "Normal")
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)


...

        Public Shared Sub FillAdaptiveParams(baseOrder As Order, priority As String)

            baseOrder.AlgoStrategy = "Adaptive"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("adaptivePriority", priority))
        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

```
...

    AvailableAlgoParams::FillAdaptiveParams(baseOrder, "Normal");
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);


...

void AvailableAlgoParams::FillAdaptiveParams(Order& baseOrder, std::string priority){
    baseOrder.algoStrategy = "Adaptive";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("adaptivePriority", priority));
    baseOrder.algoParams->push_back(tag1);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

```
...

        AvailableAlgoParams.FillAdaptiveParams(baseOrder, "Normal")
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)


...

    @staticmethod
    def FillAdaptiveParams(baseOrder: Order, priority: str):
        baseOrder.algoStrategy = "Adaptive"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("adaptivePriority", priority))
```

10.2.9.0.0.2 ArrivalPrice

The `Arrival Price` algorithmic order type will attempt to achieve, over the course of the order, the bid/ask midpoint at the time the order is submitted. The Arrival Price algo is designed to keep hidden orders that will impact a high percentage of the average daily volume (ADV). The pace of execution is determined by the user-assigned level of risk aversion and the user-defined target percent of average daily volume. How quickly the order is submitted during the day is determined by the level of urgency: the higher the urgency the faster it will execute but will expose it to a greater market impact. Market impact can be lessened by assigning lesser urgency, which is likely to lengthen the duration of the order. The user can set the max percent of ADV from 1 to 50%. The order entry screen allows the user to determine when the order will start and end regardless of whether or not the full amount of the order has been filled. By checking the box marked Allow trading past end time the algo will continue to work past the specified end time in an effort to fill the remaining portion of the order.

Parameter	Description	Values
maxPctVol	Maximum percentage of ADV	0.1 (10%) - 0.5 (50%)
riskAversion	Urgency/risk aversion	Get Done, Aggressive, Neutral, Passive
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
allowPastEndTime	Allow trading past end time	1 (true) or 0 (false)
forceCompletion	Attempt completion by the end of the day	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillArrivalPriceParams(baseOrder, 0.1, "Aggressive", "09:00:00 CET", "
        16:00:00 CET", true, true, 100000);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
          public static void FillArrivalPriceParams(Order baseOrder, double maxPctVol, string riskAversion,
        string startTime, string endTime,
            bool forceCompletion, bool allowPastTime, double monetaryValue)
          {
            baseOrder.AlgoStrategy = "ArrivalPx";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("maxPctVol", maxPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("riskAversion", riskAversion));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("allowPastEndTime", allowPastTime ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
          }
```

-
```
          Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
          AvailableAlgoParams.FillArrivalPriceParams(baseOrder, 0.1, "Aggressive", "09:00:00 CET", "16:00:00
        CET", true, true, 100000);
          client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillArrivalPriceParams(Order baseOrder, double maxPctVol, String riskAversion,
      String startTime,
            String endTime, boolean forceCompletion, boolean allowPastTime, double monetaryValue) {

        baseOrder.algoStrategy("ArrivalPx");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("maxPctVol", String.valueOf(maxPctVol)));
        baseOrder.algoParams().add(new TagValue("riskAversion", riskAversion));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("endTime", endTime));
        baseOrder.algoParams().add(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("allowPastEndTime", allowPastTime ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));

    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillArrivalPriceParams(baseOrder, 0.10000000000000001, "Aggressive", "09:00:00
       CET", "16:00:00 CET", True, True, 100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillArrivalPriceParams(baseOrder As Order, maxPctVol As Double, riskAversion As
       String, startTime As String, endTime As String,
            forceCompletion As Boolean, allowPastTime As Boolean, monetaryValue As Double)

            baseOrder.AlgoStrategy = "ArrivalPx"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("maxPctVol", maxPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("riskAversion", riskAversion))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("forceCompletion", BooleantoString(forceCompletion)))
            baseOrder.AlgoParams.Add(New TagValue("allowPastEndTime", BooleantoString(allowPastTime)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))
        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillArrivalPriceParams(baseOrder, 0.1, "Aggressive", "09:00:00 CET", "16:00:00 CET
      ", true, true, 100000);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillArrivalPriceParams(Order& baseOrder, double maxPctVol, std::string
      riskAversion, std::string startTime, std::string endTime,
    bool forceCompletion, bool allowPastTime, double monetaryValue){
    baseOrder.algoStrategy = "ArrivalPx";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("maxPctVol", std::to_string(maxPctVol)));
    TagValueSPtr tag2(new TagValue("riskAversion", riskAversion));
    TagValueSPtr tag3(new TagValue("startTime", startTime));
    TagValueSPtr tag4(new TagValue("endTime", endTime));
    TagValueSPtr tag5(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));
    TagValueSPtr tag6(new TagValue("allowPastEndTime", allowPastTime ? "1" : "0"));
    TagValueSPtr tag7(new TagValue("monetaryValue", std::to_string(monetaryValue)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
    baseOrder.algoParams->push_back(tag6);
    baseOrder.algoParams->push_back(tag7);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillArrivalPriceParams(baseOrder, 0.1,
                                                   "Aggressive", "09:00:00 CET", "16:00:00 CET", True, True
      , 100000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillArrivalPriceParams(baseOrder: Order, maxPctVol: float,
                               riskAversion: str, startTime: str, endTime: str,
                               forceCompletion: bool, allowPastTime: bool,
                               monetaryValue: float):
        baseOrder.algoStrategy = "ArrivalPx"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("maxPctVol", maxPctVol))
        baseOrder.algoParams.append(TagValue("riskAversion", riskAversion))
        baseOrder.algoParams.append(TagValue("startTime", startTime))
        baseOrder.algoParams.append(TagValue("endTime", endTime))
        baseOrder.algoParams.append(TagValue("forceCompletion",
                                             int(forceCompletion)))
        baseOrder.algoParams.append(TagValue("allowPastEndTime",
                                             int(allowPastTime)))
        baseOrder.algoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.3 Close Price

Investors submitting market or limit orders into the closing auction may adversely affect the closing price, especially when the size of the order is large relative to the average close auction volume. In order to help investors attempting to execute towards the end of the trading session we have developed the `Close Price` algo Strategy. This algo breaks down large order amounts and determines the timing of order entry so that it will continuously execute in order to minimize slippage. The start and pace of execution are determined by the user who assigns a level of market risk and specifies the target percentage of volume, while the algo considers the prior volatility of the stock.

Parameter	Description	Values
maxPctVol	Maximum percentage of ADV	0.1 (10%) - 0.5 (50%)
riskAversion	Urgency/risk aversion	Get Done, Aggressive, Neutral, Passive
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
forceCompletion	Attempt completion by the end of the day	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillClosePriceParams(baseOrder, 0.5, "Neutral", "12:00:00 EST", true, 10000
        0);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
          public static void FillClosePriceParams(Order baseOrder, double maxPctVol, string riskAversion,
        string startTime,
            bool forceCompletion, double monetaryValue)
          {
            baseOrder.AlgoStrategy = "ClosePx";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("maxPctVol", maxPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("riskAversion", riskAversion));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
          }
```

-
```
          Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillClosePriceParams(baseOrder, 0.5, "Neutral", "12:00:00 EST", true, 100000);
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillClosePriceParams(Order baseOrder, double maxPctVol, String riskAversion, String
      startTime,
            boolean forceCompletion, double monetaryValue){

        baseOrder.algoStrategy("ClosePx");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("maxPctVol", String.valueOf(maxPctVol)));
        baseOrder.algoParams().add(new TagValue("riskAversion", riskAversion));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));
    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillClosePriceParams(baseOrder, 0.5, "Neutral", "12:00:00 EST", True, 100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillClosePriceParams(baseOrder As Order, maxPctVol As Double, riskAversion As
       String, startTime As String, forceCompletion As Boolean, monetaryValue As Double)

            baseOrder.AlgoStrategy = "ClosePx"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("maxPctVol", maxPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("riskAversion", riskAversion))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("forceCompletion", BooleantoString(forceCompletion)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))

        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillClosePriceParams(baseOrder, 0.5, "Neutral", "12:00:00 EST", true, 100000);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillClosePriceParams(Order& baseOrder, double maxPctVol, std::string riskAversion
      , std::string startTime, bool forceCompletion, double monetaryValue){
    baseOrder.algoStrategy = "ClosePx";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("maxPctVol", std::to_string(maxPctVol)));
    TagValueSPtr tag2(new TagValue("riskAversion",riskAversion));
    TagValueSPtr tag3(new TagValue("startTime", startTime));
    TagValueSPtr tag4(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));
    TagValueSPtr tag5(new TagValue("monetaryValue", std::to_string(monetaryValue)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillClosePriceParams(baseOrder, 0.5, "Neutral",
                                                 "12:00:00 EST", True, 100000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillClosePriceParams(baseOrder: Order, maxPctVol: float, riskAversion: str,
                             startTime: str, forceCompletion: bool,
                             monetaryValue: float):
        baseOrder.AlgoStrategy = "ClosePx"
        baseOrder.AlgoParams = []
        baseOrder.AlgoParams.append(TagValue("maxPctVol", maxPctVol))
        baseOrder.AlgoParams.append(TagValue("riskAversion", riskAversion))
        baseOrder.AlgoParams.append(TagValue("startTime", startTime))
        baseOrder.AlgoParams.append(TagValue("forceCompletion", int(forceCompletion)))
        baseOrder.AlgoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.4 DarkIce

The `Dark Ice` order type develops the concept of privacy adopted by orders such as Iceberg or Reserve, using a proprietary algorithm to further hide the volume displayed to the market by the order. Clients can determine the timeframe an order remains live and have the option to allow trading past end time in the event it is unfilled by the stated end time. In order to minimize market impact in the event of large orders, users can specify a display size to be shown to the market different from the actual order size. Additionally, the Dark Ice algo randomizes the display size +/- 50% based upon the probability of the price moving favorably. Further, using calculated probabilities, the algo decides whether to place the order at the limit price or one tick lower than the current offer for buy orders and one tick higher than the current bid for sell orders.

Parameter	Description	Values
displaySize	Order size to be displayed	
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
allowPastEndTime	Allow trading past end time	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillDarkIceParams(baseOrder, 10, "09:00:00 CET", "16:00:00 CET", true, 1000
        00);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
          public static void FillDarkIceParams(Order baseOrder, int displaySize, string startTime, string
        endTime,
            bool allowPastEndTime, double monetaryValue)
          {
            baseOrder.AlgoStrategy = "DarkIce";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("displaySize", displaySize.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
          }
```

-
```
          Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
          AvailableAlgoParams.FillDarkIceParams(baseOrder, 10, "09:00:00 CET", "16:00:00 CET", true, 100000);
          client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillDarkIceParams(Order baseOrder, int displaySize, String startTime, String endTime
      ,
            boolean allowPastEndTime, double monetaryValue) {

        baseOrder.algoStrategy("DarkIce");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("displaySize", String.valueOf(displaySize)));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("endTime", endTime));
        baseOrder.algoParams().add(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));

    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillDarkIceParams(baseOrder, 10, "09:00:00 CET", "16:00:00 CET", True, 100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillDarkIceParams(baseOrder As Order, displaySize As Integer, startTime As
       String, endTime As String,
            allowPastEndTime As Boolean, monetaryValue As Double)

            baseOrder.AlgoStrategy = "DarkIce"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("displaySize", displaySize.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("allowPastEndTime", BooleantoString(allowPastEndTime)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))
        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillDarkIceParams(baseOrder, 10, "09:00:00 CET", "16:00:00 CET", true, 100000);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillDarkIceParams(Order& baseOrder, int displaySize, std::string startTime,
      std::string endTime, bool allowPastEndTime, double monetaryValue){
    baseOrder.algoStrategy = "DarkIce";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("displaySize", std::to_string(displaySize)));
    TagValueSPtr tag2(new TagValue("startTime", startTime));
    TagValueSPtr tag3(new TagValue("endTime", endTime));
    TagValueSPtr tag4(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
    TagValueSPtr tag5(new TagValue("monetaryValue", std::to_string(monetaryValue)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillDarkIceParams(baseOrder, 10,
                                              "09:00:00 CET", "16:00:00 CET", True, 100000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillDarkIceParams(baseOrder: Order, displaySize: int, startTime: str,
                          endTime: str, allowPastEndTime: bool,
                          monetaryValue: float):
        baseOrder.algoStrategy = "DarkIce"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("displaySize", displaySize))
        baseOrder.algoParams.append(TagValue("startTime", startTime))
        baseOrder.algoParams.append(TagValue("endTime", endTime))
        baseOrder.algoParams.append(TagValue("allowPastEndTime",
                                             int(allowPastEndTime)))
        baseOrder.algoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.5 Accumulate/Distribute

The `Accumulate/Distribute` algo can help you to achieve the best price for a large volume order without being noticed in the market, and can be set up for high frequency trading. By slicing your order into smaller randomly-sized order increments that are released at random time intervals within a user-defined time period, the algo allows the trading of large blocks of stock and other instruments without being detected in the market. The algo allows limit, market, and relative order types. It is important to keep in mind the API A/D algo will not have all available parameters of the A/D algos that can be created in TWS.

Parameter	Description	Values
componentSize	Quantity of increment	Cannot exceed initial size
timeBetweenOrders	Time interval in seconds between each order	
randomizeTime20	Randomise time period by +/- 20%	1 (true) or 0 (false)
randomizeSize55	Randomise size by +/- 55%	1 (true) or 0 (false)
giveUp	Number associated with the clearing	
catchUp	Catch up in time	1 (true) or 0 (false)
waitForFill	Wait for current order to fill before submitting next order	1 (true) or 0 (false)
startTime	Algorithm starting time	YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	YYYYMMDD-hh:mm:ss TMZ

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            // The Time Zone in "startTime" and "endTime" attributes is ignored and always defaulted to GMT
            AvailableAlgoParams.FillAccumulateDistributeParams(baseOrder, 10, 60, true, true, 1, true, true
, "20161010-12:00:00 GMT", "20161010-16:00:00 GMT");
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
        public static void FillAccumulateDistributeParams(Order baseOrder, int componentSize, int
timeBetweenOrders, bool randomizeTime20, bool randomizeSize55,
            int giveUp, bool catchUp, bool waitForFill, string startTime, string endTime)
        {
            baseOrder.AlgoStrategy = "AD";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("componentSize", componentSize.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("timeBetweenOrders", timeBetweenOrders.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("randomizeTime20", randomizeTime20 ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("randomizeSize55", randomizeSize55 ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("giveUp", giveUp.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("catchUp", catchUp ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("waitForFill", waitForFill ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        // The Time Zone in "startTime" and "endTime" attributes is ignored and always defaulted to GMT
        AvailableAlgoParams.FillAccumulateDistributeParams(baseOrder, 10, 60, true, true, 1, true, true, "
20161010-12:00:00 GMT", "20161010-16:00:00 GMT");
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
        public static void FillAccumulateDistributeParams(Order baseOrder, int componentSize, int
          timeBetweenOrders, boolean randomizeTime20, boolean randomizeSize55,
                int giveUp, boolean catchUp, boolean waitForFill, String startTime, String endTime) {

            baseOrder.algoStrategy("AD");
            baseOrder.algoParams(new ArrayList<>());
            baseOrder.algoParams().add(new TagValue("componentSize", String.valueOf(componentSize)));
            baseOrder.algoParams().add(new TagValue("timeBetweenOrders", String.valueOf(timeBetweenOrders)));
            baseOrder.algoParams().add(new TagValue("randomizeTime20", randomizeTime20 ? "1" : "0"));
            baseOrder.algoParams().add(new TagValue("randomizeSize55", randomizeSize55 ? "1" : "0"));
            baseOrder.algoParams().add(new TagValue("giveUp", String.valueOf(giveUp)));
            baseOrder.algoParams().add(new TagValue("catchUp", catchUp ? "1" : "0"));
            baseOrder.algoParams().add(new TagValue("waitForFill", waitForFill ? "1" : "0"));
            baseOrder.algoParams().add(new TagValue("startTime", startTime));
            baseOrder.algoParams().add(new TagValue("endTime", endTime));

        }
```

-
```
            Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
            ' The Time Zone in "startTime" and "endTime" attributes is ignored and always defaulted to GMT
            AvailableAlgoParams.FillAccumulateDistributeParams(baseOrder, 10, 60, True, True, 1, True, True,
           "20161010-12:00:00 GMT", "20161010-16:00:00 GMT")
            client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
            Public Shared Sub FillAccumulateDistributeParams(baseOrder As Order, componentSize As Integer,
           timeBetweenOrders As Integer, randomizeTime20 As Boolean, randomizeSize55 As Boolean,
                giveUp As Integer, catchUp As Boolean, waitForFill As Boolean, startTime As String, endTime As
           String)

                baseOrder.AlgoStrategy = "AD"
                baseOrder.AlgoParams = New List(Of TagValue)
                baseOrder.AlgoParams.Add(New TagValue("componentSize", componentSize.ToString()))
                baseOrder.AlgoParams.Add(New TagValue("timeBetweenOrders", timeBetweenOrders.ToString()))
                baseOrder.AlgoParams.Add(New TagValue("randomizeTime20", BooleantoString(randomizeTime20)))
                baseOrder.AlgoParams.Add(New TagValue("randomizeSize55", BooleantoString(randomizeSize55)))
                baseOrder.AlgoParams.Add(New TagValue("giveUp", giveUp.ToString()))
                baseOrder.AlgoParams.Add(New TagValue("catchUp", BooleantoString(catchUp)))
                baseOrder.AlgoParams.Add(New TagValue("waitForFill", BooleantoString(waitForFill)))
                baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
                baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            End Sub
```

-
```
        Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
        // The Time Zone in "startTime" and "endTime" attributes is ignored and always defaulted to GMT
        AvailableAlgoParams::FillAccumulateDistributeParams(baseOrder, 10, 60, true, true, 1, true, true, "
          20161010-12:00:00 GMT", "20161010-16:00:00 GMT");
        m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
    void AvailableAlgoParams::FillAccumulateDistributeParams(Order& baseOrder, int componentSize, int
          timeBetweenOrders, bool randomizeTime20, bool randomizeSize55,
        int giveUp, bool catchUp, bool waitForFill, std::string startTime, std::string endTime){
        baseOrder.algoStrategy = "AD";
        baseOrder.algoParams.reset(new TagValueList());
        TagValueSPtr tag1(new TagValue("componentSize", std::to_string(componentSize)));
        TagValueSPtr tag2(new TagValue("timeBetweenOrders",std::to_string(timeBetweenOrders)));
        TagValueSPtr tag3(new TagValue("randomizeTime20", randomizeTime20 ? "1" : "0"));
        TagValueSPtr tag4(new TagValue("randomizeSize55", randomizeSize55 ? "1" : "0"));
        TagValueSPtr tag5(new TagValue("giveUp", std::to_string(giveUp)));
        TagValueSPtr tag6(new TagValue("catchUp", catchUp ? "1" : "0"));
        TagValueSPtr tag7(new TagValue("waitForFill", waitForFill ? "1" : "0"));
        TagValueSPtr tag8(new TagValue("startTime", startTime));
        TagValueSPtr tag9(new TagValue("endTime", endTime));
        baseOrder.algoParams->push_back(tag1);
        baseOrder.algoParams->push_back(tag2);
        baseOrder.algoParams->push_back(tag3);
        baseOrder.algoParams->push_back(tag4);
        baseOrder.algoParams->push_back(tag5);
        baseOrder.algoParams->push_back(tag6);
        baseOrder.algoParams->push_back(tag7);
        baseOrder.algoParams->push_back(tag8);
        baseOrder.algoParams->push_back(tag9);
    }
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        # The Time Zone in "startTime" and "endTime" attributes is ignored and always defaulted to GMT
        AvailableAlgoParams.FillAccumulateDistributeParams(baseOrder, 10, 60,
                                                           True, True, 1, True, True,
                                                           "20161010-12:00:00 GMT", "20161010-16:00:00 GMT"
    )
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillAccumulateDistributeParams(baseOrder: Order, componentSize: int,
                                       timeBetweenOrders: int, randomizeTime20: bool, randomizeSize55: bool
      ,
                                       giveUp: int, catchUp: bool, waitForFill: bool, startTime: str,
                                       endTime: str):
        baseOrder.algoStrategy = "AD"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("componentSize", componentSize))
        baseOrder.algoParams.append(TagValue("timeBetweenOrders", timeBetweenOrders))
        baseOrder.algoParams.append(TagValue("randomizeTime20",
                                             int(randomizeTime20)))
        baseOrder.algoParams.append(TagValue("randomizeSize55",
                                             int(randomizeSize55)))
        baseOrder.algoParams.append(TagValue("giveUp", giveUp))
        baseOrder.algoParams.append(TagValue("catchUp", int(catchUp)))
        baseOrder.algoParams.append(TagValue("waitForFill", int(waitForFill)))
        baseOrder.algoParams.append(TagValue("startTime", startTime))
        baseOrder.algoParams.append(TagValue("endTime", endTime))
```

10.2.9.0.0.6 Percentage of Volume

The `Percent of Volume` algo can limit the contribution of orders to overall average daily volume in order to minimize impact. Clients can set a value between 1-50% to control their participation between stated start and end times. Order quantity and volume distribution over the day is determined using the target percent of volume you entered along with continuously updated volume forecasts calculated from TWS market data. In addition, the algo can be set to avoid taking liquidity, which may help avoid liquidity-taker fees and could result in liquidity-adding rebates. By checking the Attempt to never take liquidity box, the algo is discouraged from hitting the bid or lifting the offer if possible. However, this may also result in greater deviations from the benchmark, and in partial fills, since the posted bid/offer may not always get hit as the price moves up/down. IB will use best efforts not to take liquidity when this box is checked, however, there will be times that it cannot be avoided.

Parameter	Description	Values
pctVol	Target Percentage	0.1 (10%) - 0.5 (50%)
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
noTakeLiq	Attempt to never take liquidity	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillPctVolParams(baseOrder, 0.5, "12:00:00 EST", "14:00:00 EST", true, 1000
    00);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
        public static void FillPctVolParams(Order baseOrder, double pctVol, string startTime, string
      endTime, bool noTakeLiq, double monetaryValue)
        {
            baseOrder.AlgoStrategy = "PctVol";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("pctVol", pctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillPctVolParams(baseOrder, 0.5, "12:00:00 EST", "14:00:00 EST", true, 100000);
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillPctVolParams(Order baseOrder, double pctVol, String startTime, String endTime,
      boolean noTakeLiq, double monetaryValue) {

        baseOrder.algoStrategy("PctVol");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("pctVol", String.valueOf(pctVol)));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("endTime", endTime));
        baseOrder.algoParams().add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));
    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillPctVolParams(baseOrder, 0.5, "12:00:00 EST", "14:00:00 EST", True, 100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillPctVolParams(baseOrder As Order, pctVol As Double, startTime As String,
       endTime As String, noTakeLiq As Boolean, monetaryValue As Double)

            baseOrder.AlgoStrategy = "PctVol"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("pctVol", pctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("noTakeLiq", BooleantoString(noTakeLiq)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))

        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillPctVolParams(baseOrder, 0.5, "12:00:00 EST", "14:00:00 EST", true, 100000);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillPctVolParams(Order& baseOrder, double pctVol, std::string startTime,
      std::string endTime, bool noTakeLiq, double monetaryValue){
    baseOrder.algoStrategy = "PctVol";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("pctVol", std::to_string(pctVol)));
    TagValueSPtr tag2(new TagValue("startTime", startTime));
    TagValueSPtr tag3(new TagValue("endTime", endTime));
    TagValueSPtr tag4(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
    TagValueSPtr tag5(new TagValue("monetaryValue", std::to_string(monetaryValue)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillPctVolParams(baseOrder, 0.5,
                                             "12:00:00 EST", "14:00:00 EST", True, 100000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillPctVolParams(baseOrder: Order, pctVol: float, startTime: str,
                         endTime: str, noTakeLiq: bool,
                         monetaryValue: float):
        baseOrder.algoStrategy = "PctVol"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("pctVol", pctVol))
        baseOrder.algoParams.append(TagValue("startTime", startTime))
        baseOrder.algoParams.append(TagValue("endTime", endTime))
        baseOrder.algoParams.append(TagValue("noTakeLiq", int(noTakeLiq)))
        baseOrder.algoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.7 TWAP

The TWAP algo aims to achieve the time-weighted average price calculated from the time you submit the order to the time it completes. Incomplete orders at the end of the stated completion time will continue to fill if the box 'allow trading past end time' is checked. Users can set the order to trade only when specified conditions are met. Those user-defined inputs include when the order is marketable, when the midpoint matches the required price, when the same side (buy or sell) matches to make the order marketable or when the last traded price would make the order marketable. For the TWAP algo, the average price calculation is calculated from the order entry time through the close of the market and will only attempt to execute when the criterion is met. The order may not fill throughout its stated duration and so the order is not guaranteed. TWAP is available for all US equities.

Parameter	Description	Values
strategyType	Trade strategy	Marketable, Matching, Midpoint, Matching Same Side, Matching Last
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
allowPastEndTime	Allow trading past end time	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillTwapParams(baseOrder, "Marketable", "09:00:00 CET", "16:00:00 CET",
      true, 100000);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
        public static void FillTwapParams(Order baseOrder, string strategyType, string startTime, string
      endTime, bool allowPastEndTime, double monetaryValue)
        {
            baseOrder.AlgoStrategy = "Twap";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("strategyType", strategyType));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillTwapParams(baseOrder, "Marketable", "09:00:00 CET", "16:00:00 CET", true, 1
      00000);
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillTwapParams(Order baseOrder, String strategyType, String startTime, String
      endTime,
            boolean allowPastEndTime, double monetaryValue) {

        baseOrder.algoStrategy("Twap");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("strategyType", strategyType));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("endTime", endTime));
        baseOrder.algoParams().add(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));

    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillTwapParams(baseOrder, "Marketable", "09:00:00 CET", "16:00:00 CET", True,
      100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillTwapParams(baseOrder As Order, strategyType As String, startTime As String,
      endTime As String,
            allowPastEndTime As Boolean, monetaryValue As Double)

            baseOrder.AlgoStrategy = "Twap"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("strategyType", strategyType))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("allowPastEndTime", BooleantoString(allowPastEndTime)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))

        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillTwapParams(baseOrder, "Marketable", "09:00:00 CET", "16:00:00 CET", true, 1000
      00);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillTwapParams(Order& baseOrder, std::string strategyType, std::string startTime,
       std::string endTime, bool allowPastEndTime, double monetaryValue){
    baseOrder.algoStrategy = "Twap";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("strategyType", strategyType));
    TagValueSPtr tag2(new TagValue("startTime", startTime));
    TagValueSPtr tag3(new TagValue("endTime", endTime));
    TagValueSPtr tag4(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
    TagValueSPtr tag5(new TagValue("monetaryValue", std::to_string(monetaryValue)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillTwapParams(baseOrder, "Marketable",
                                           "09:00:00 CET", "16:00:00 CET", True, 100000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillTwapParams(baseOrder: Order, strategyType: str, startTime: str,
                       endTime: str, allowPastEndTime: bool,
                       monetaryValue: float):
        baseOrder.algoStrategy = "Twap"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("strategyType", strategyType))
        baseOrder.algoParams.append(TagValue("startTime", startTime))
        baseOrder.algoParams.append(TagValue("endTime", endTime))
        baseOrder.algoParams.append(TagValue("allowPastEndTime",
                                             int(allowPastEndTime)))
        baseOrder.algoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.8 Price Variant Percentage of Volume Strategy

`Price Variant Percentage of Volume Strategy` - This algo allows you to participate in volume at a user-defined rate that varies over time depending on the market price of the security. This algo allows you to buy more aggressively when the price is low and be more passive as the price increases, and just the opposite for sell orders. The order quantity and volume distribution over the time during which the order is active is determined using the target percent of volume you entered along with continuously updated volume forecasts calculated from TWS market data.

Parameter	Description	Values
pctVol	Target Percentage	0.1 (10%) - 0.5 (50%)
deltaPctVol	Target Percentage Change Rate	0.1 (10%) - 0.5 (50%)
minPctVol4Px	Minimum Target Percentage	0.1 (10%) - 0.5 (50%)
maxPctVol4Px	Maximum Target Percentage	0.1 (10%) - 0.5 (50%)
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
noTakeLiq	Attempt to never take liquidity	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillPriceVariantPctVolParams(baseOrder, 0.1, 0.05, 0.01, 0.2, "12:00:00 EST
      ", "14:00:00 EST", true, 100000);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
        public static void FillPriceVariantPctVolParams(Order baseOrder, double pctVol, double deltaPctVol,
       double minPctVol4Px,
            double maxPctVol4Px, String startTime, String endTime, bool noTakeLiq, double monetaryValue)
        {
            baseOrder.AlgoStrategy = "PctVolPx";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("pctVol", pctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("deltaPctVol", deltaPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("minPctVol4Px", minPctVol4Px.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("maxPctVol4Px", maxPctVol4Px.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
        }
```

```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillPriceVariantPctVolParams(baseOrder, 0.1, 0.05, 0.01, 0.2, "12:00:00 EST", "
      14:00:00 EST", true, 100000);
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillPriceVariantPctVolParams(Order baseOrder, double pctVol, double deltaPctVol,
      double minPctVol4Px,
            double maxPctVol4Px, String startTime, String endTime, boolean noTakeLiq, double monetaryValue)
      {

        baseOrder.algoStrategy("PctVolPx");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("pctVol", String.valueOf(pctVol)));
        baseOrder.algoParams().add(new TagValue("deltaPctVol", String.valueOf(deltaPctVol)));
        baseOrder.algoParams().add(new TagValue("minPctVol4Px", String.valueOf(minPctVol4Px)));
        baseOrder.algoParams().add(new TagValue("maxPctVol4Px", String.valueOf(maxPctVol4Px)));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("endTime", endTime));
        baseOrder.algoParams().add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));
    }
```

```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillPriceVariantPctVolParams(baseOrder, 0.10000000000000001,
       0.050000000000000003, 0.01, 0.20000000000000001, "12:00:00 EST", "14:00:00 EST", True, 100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillPriceVariantPctVolParams(baseOrder As Order, pctVol As Double, deltaPctVol As
       Double, minPctVol4Px As Double,
            maxPctVol4Px As Double, startTime As String, endTime As String, noTakeLiq As Boolean,
       monetaryValue As Double)

            baseOrder.AlgoStrategy = "PctVolPx"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("pctVol", pctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("deltaPctVol", deltaPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("minPctVol4Px", minPctVol4Px.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("maxPctVol4Px", maxPctVol4Px.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("noTakeLiq", BooleantoString(noTakeLiq)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))

        End Sub
```

```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillPriceVariantPctVolParams(baseOrder, 0.1, 0.05, 0.01, 0.2, "12:00:00 EST", "
      14:00:00 EST", true, 100000);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
    void AvailableAlgoParams::FillPriceVariantPctVolParams(Order baseOrder, double pctVol, double deltaPctVol,
          double minPctVol4Px,
                                                           double maxPctVol4Px, std::string startTime,
      std::string endTime, bool noTakeLiq, double monetaryValue){
    baseOrder.algoStrategy = "PctVolPx";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("pctVol", std::to_string(pctVol)));
    TagValueSPtr tag2(new TagValue("deltaPctVol", std::to_string(deltaPctVol)));
    TagValueSPtr tag3(new TagValue("minPctVol4Px", std::to_string(minPctVol4Px)));
    TagValueSPtr tag4(new TagValue("maxPctVol4Px", std::to_string(maxPctVol4Px)));
    TagValueSPtr tag5(new TagValue("startTime", startTime));
    TagValueSPtr tag6(new TagValue("endTime", endTime));
    TagValueSPtr tag7(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
    TagValueSPtr tag8(new TagValue("monetaryValue", std::to_string(monetaryValue)));
```

```
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
    baseOrder.algoParams->push_back(tag6);
    baseOrder.algoParams->push_back(tag7);
    baseOrder.algoParams->push_back(tag8);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillPriceVariantPctVolParams(baseOrder,
                                                         0.1, 0.05, 0.01, 0.2, "12:00:00 EST", "14:00:00
    EST", True,
                                                         100000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillPriceVariantPctVolParams(baseOrder: Order, pctVol: float,
                                     deltaPctVol: float, minPctVol4Px: float,
                                     maxPctVol4Px: float, startTime: str,
                                     endTime: str, noTakeLiq: bool,
                                     monetaryValue: float):
        baseOrder.AlgoStrategy = "PctVolPx"
        baseOrder.AlgoParams = []
        baseOrder.AlgoParams.append(TagValue("pctVol", pctVol))
        baseOrder.AlgoParams.append(TagValue("deltaPctVol", deltaPctVol))
        baseOrder.AlgoParams.append(TagValue("minPctVol4Px", minPctVol4Px))
        baseOrder.AlgoParams.append(TagValue("maxPctVol4Px", maxPctVol4Px))
        baseOrder.AlgoParams.append(TagValue("startTime", startTime))
        baseOrder.AlgoParams.append(TagValue("endTime", endTime))
        baseOrder.AlgoParams.append(TagValue("noTakeLiq", int(noTakeLiq)))
        baseOrder.AlgoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.9 Size Variant Percentage of Volume Strategy

`Size Variant Percentage of Volume Strategy` - This algo allows you to participate in volume at a user-defined rate that varies over time depending on the remaining size of the order. Define the target percent rate at the start time (Initial Participation Rate) and at the end time (Terminal Participation Rate), and the algo calculates the participation rate over time between the two based on the remaining order size. This allows the order to be more aggressive initially and less aggressive toward the end, or vice versa.

Parameter	Description	Values
startPctVol	Initial Target Percentage	0.1 (10%) - 0.5 (50%)
endPctVol	Terminal Target Percentage	0.1 (10%) - 0.5 (50%)
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
noTakeLiq	Attempt to never take liquidity	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillSizeVariantPctVolParams(baseOrder, 0.2, 0.4, "12:00:00 EST", "14:00:00
    EST", true, 100000);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
        public static void FillSizeVariantPctVolParams(Order baseOrder, double startPctVol, double
      endPctVol,
            String startTime, String endTime, bool noTakeLiq, double monetaryValue)
        {
            baseOrder.AlgoStrategy = "PctVolSz";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("startPctVol", startPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("endPctVol", endPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillSizeVariantPctVolParams(baseOrder, 0.2, 0.4, "12:00:00 EST", "14:00:00 EST"
      , true, 100000);
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillSizeVariantPctVolParams(Order baseOrder, double startPctVol, double endPctVol,
            String startTime, String endTime, boolean noTakeLiq, double monetaryValue){

        baseOrder.algoStrategy("PctVolSz");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("startPctVol", String.valueOf(startPctVol)));
        baseOrder.algoParams().add(new TagValue("endPctVol", String.valueOf(endPctVol)));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("endTime", endTime));
        baseOrder.algoParams().add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));
    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillSizeVariantPctVolParams(baseOrder, 0.20000000000000001,
       0.40000000000000002, "12:00:00 EST", "14:00:00 EST", True, 100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillSizeVariantPctVolParams(baseOrder As Order, startPctVol As Double, endPctVol
       As Double,
            startTime As String, endTime As String, noTakeLiq As Boolean, monetaryValue As Double)

            baseOrder.AlgoStrategy = "PctVolSz"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("startPctVol", startPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("endPctVol", endPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("noTakeLiq", BooleantoString(noTakeLiq)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))

        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillSizeVariantPctVolParams(baseOrder, 0.2, 0.4, "12:00:00 EST", "14:00:00 EST",
      true, 100000);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillSizeVariantPctVolParams(Order baseOrder, double startPctVol, double endPctVol
      ,
                                std::string startTime, std::string endTime, bool noTakeLiq, double
      monetaryValue){
    baseOrder.algoStrategy = "PctVolSz";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("startPctVol", std::to_string(startPctVol)));
    TagValueSPtr tag2(new TagValue("endPctVol", std::to_string(endPctVol)));
    TagValueSPtr tag3(new TagValue("startTime", startTime));
    TagValueSPtr tag4(new TagValue("endTime", endTime));
    TagValueSPtr tag5(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
    TagValueSPtr tag6(new TagValue("monetaryValue", std::to_string(monetaryValue)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
    baseOrder.algoParams->push_back(tag6);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillSizeVariantPctVolParams(baseOrder,
                                                        0.2, 0.4, "12:00:00 EST", "14:00:00 EST", True, 100
      000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillSizeVariantPctVolParams(baseOrder: Order, startPctVol: float,
                                    endPctVol: float, startTime: str,
                                    endTime: str, noTakeLiq: bool,
                                    monetaryValue: float):
        baseOrder.AlgoStrategy = "PctVolSz"
        baseOrder.AlgoParams = []
        baseOrder.AlgoParams.append(TagValue("startPctVol", startPctVol))
        baseOrder.AlgoParams.append(TagValue("endPctVol", endPctVol))
        baseOrder.AlgoParams.append(TagValue("startTime", startTime))
        baseOrder.AlgoParams.append(TagValue("endTime", endTime))
        baseOrder.AlgoParams.append(TagValue("noTakeLiq", int(noTakeLiq)))
        baseOrder.AlgoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.10 Time Variant Percentage of Volume Strategy

`Time Variant Percentage of Volume Strategy` - This algo allows you to participate in volume at a user-defined rate that varies with time. Define the target percent rate at the start time and at the end time, and the algo calculates the participation rate over time between the two. This allows the order to be more aggressive initially and less aggressive toward the end, or vice versa.

Parameter	Description	Values
startPctVol	Initial Target Percentage	0.1 (10%) - 0.5 (50%)
endPctVol	Terminal Target Percentage	0.1 (10%) - 0.5 (50%)
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
noTakeLiq	Attempt to never take liquidity	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillTimeVariantPctVolParams(baseOrder, 0.2, 0.4, "12:00:00 EST", "14:00:00
     EST", true, 100000);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
        public static void FillTimeVariantPctVolParams(Order baseOrder, double startPctVol, double
      endPctVol,
            String startTime, String endTime, bool noTakeLiq, double monetaryValue)
        {
            baseOrder.AlgoStrategy = "PctVolTm";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("startPctVol", startPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("endPctVol", endPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillTimeVariantPctVolParams(baseOrder, 0.2, 0.4, "12:00:00 EST", "14:00:00 EST"
      , true, 100000);
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillTimeVariantPctVolParams(Order baseOrder, double startPctVol, double endPctVol,
            String startTime, String endTime, boolean noTakeLiq, double monetaryValue){

        baseOrder.algoStrategy("PctVolTm");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("startPctVol", String.valueOf(startPctVol)));
        baseOrder.algoParams().add(new TagValue("endPctVol", String.valueOf(endPctVol)));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("endTime", endTime));
        baseOrder.algoParams().add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));
    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillTimeVariantPctVolParams(baseOrder, 0.20000000000000001,
      0.40000000000000002, "12:00:00 EST", "14:00:00 EST", True, 100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillTimeVariantPctVolParams(baseOrder As Order, startPctVol As Double, endPctVol
      As Double,
            startTime As String, endTime As String, noTakeLiq As Boolean, monetaryValue As Double)

            baseOrder.AlgoStrategy = "PctVolTm"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("startPctVol", startPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("endPctVol", endPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("noTakeLiq", BooleantoString(noTakeLiq)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))

        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillTimeVariantPctVolParams(baseOrder, 0.2, 0.4, "12:00:00 EST", "14:00:00 EST",
      true, 100000);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillTimeVariantPctVolParams(Order baseOrder, double startPctVol, double endPctVol
      , std::string startTime,
                                std::string endTime, bool noTakeLiq, double monetaryValue){
    baseOrder.algoStrategy = "PctVolTm";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("startPctVol", std::to_string(startPctVol)));
    TagValueSPtr tag2(new TagValue("endPctVol", std::to_string(endPctVol)));
    TagValueSPtr tag3(new TagValue("startTime", startTime));
    TagValueSPtr tag4(new TagValue("endTime", endTime));
    TagValueSPtr tag5(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
    TagValueSPtr tag6(new TagValue("monetaryValue", std::to_string(monetaryValue)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
    baseOrder.algoParams->push_back(tag6);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillTimeVariantPctVolParams(baseOrder,
                                                        0.2, 0.4, "12:00:00 EST", "14:00:00 EST", True, 100
      000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillTimeVariantPctVolParams(baseOrder: Order, startPctVol: float,
                                    endPctVol: float, startTime: str,
                                    endTime: str, noTakeLiq: bool,
                                    monetaryValue: float):
        baseOrder.AlgoStrategy = "PctVolTm"
        baseOrder.AlgoParams = []
        baseOrder.AlgoParams.append(TagValue("startPctVol", startPctVol))
        baseOrder.AlgoParams.append(TagValue("endPctVol", endPctVol))
        baseOrder.AlgoParams.append(TagValue("startTime", startTime))
        baseOrder.AlgoParams.append(TagValue("endTime", endTime))
        baseOrder.AlgoParams.append(TagValue("noTakeLiq", int(noTakeLiq)))
        baseOrder.AlgoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.11 VWAP

IB's best-efforts `VWAP` algo seeks to achieve the Volume-Weighted Average price (VWAP), calculated from the time you submit the order to the close of the market.

If you require a guaranteed VWAP, please refer to IB's `Guaranteed VWAP` order type.

Best-efforts VWAP algo is a lower-cost alternative to the Guaranteed VWAP that enables the user to attempt never to take liquidity while also trading past the end time. Because the order may not be filled on the bid or at the ask prices, there is a trade-off with this algo. The order may not fully fill if the user is attempting to avoid liquidity-taking fees and/or maximize liquidity-adding rebates, and may miss the benchmark by asking to stay on the bid or ask. The user can determine the maximum percentage of average daily volume (up to 50%) his order will comprise. The system will generate the VWAP from the time the order is entered through the close of trading, and the order can be limited to trading over a pre-determined period. The user can request the order to continue beyond its stated end time if unfilled at the end of the stated period. The best-efforts VWAP algo is available for all US equities.

Parameter	Description	Values
maxPctVol	Maximum percentage of average daily volume	0.1 (10%) - 0.5 (50%)
startTime	Algorithm starting time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
endTime	Algorithm ending time	hh:mm:ss TMZ or YYYYMMDD-hh:mm:ss TMZ
allowPastEndTime	Allow trading past end time	1 (true) or 0 (false)
noTakeLiq	Attempt to never take liquidity	1 (true) or 0 (false)
speedUp	Compensate for the decreased fill rate due to presence of limit price	1 (true) or 0 (false)
monetaryValue	Cash Quantity	

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillVwapParams(baseOrder, 0.2, "09:00:00 CET", "16:00:00 CET", true, true,
        true, 100000);
            client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
        public static void FillVwapParams(Order baseOrder, double maxPctVol, string startTime, string
        endTime,
            bool allowPastEndTime, bool noTakeLiq, bool speedUp, double monetaryValue)
        {
            baseOrder.AlgoStrategy = "Vwap";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("maxPctVol", maxPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("speedUp", speedUp ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("monetaryValue", monetaryValue.ToString()));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillVwapParams(baseOrder, 0.2, "09:00:00 CET", "16:00:00 CET", true, true, true
        , 100000);
        client.placeOrder(nextOrderId++, ContractSamples.USStockAtSmart(), baseOrder);
```

...

```
    public static void FillVwapParams(Order baseOrder, double maxPctVol, String startTime, String endTime,
            boolean allowPastEndTime, boolean noTakeLiq, boolean speedUp, double monetaryValue) {

        baseOrder.algoStrategy("Vwap");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("maxPctVol", String.valueOf(maxPctVol)));
        baseOrder.algoParams().add(new TagValue("startTime", startTime));
        baseOrder.algoParams().add(new TagValue("endTime", endTime));
        baseOrder.algoParams().add(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("speedUp", speedUp ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("monetaryValue", String.valueOf(monetaryValue)));

    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillVwapParams(baseOrder, 0.20000000000000001, "09:00:00 CET", "16:00:00 CET",
        True, True, True, 100000)
        client.placeOrder(increment(nextOrderId), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
        Public Shared Sub FillVwapParams(baseOrder As Order, maxPctVol As Double, startTime As String,
        endTime As String,
            allowPastEndTime As Boolean, noTakeLiq As Boolean, speedUp As Boolean, monetaryValue As Double)

            baseOrder.AlgoStrategy = "Vwap"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("maxPctVol", maxPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("allowPastEndTime", BooleantoString(allowPastEndTime)))
            baseOrder.AlgoParams.Add(New TagValue("noTakeLiq", BooleantoString(noTakeLiq)))
            baseOrder.AlgoParams.Add(New TagValue("speedUp", BooleantoString(speedUp)))
            baseOrder.AlgoParams.Add(New TagValue("monetaryValue", monetaryValue.ToString()))

        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillBalanceImpactRiskParams(baseOrder, 0.1, "Aggressive", true);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillVwapParams(Order& baseOrder, double maxPctVol, std::string startTime,
      std::string endTime, bool allowPastEndTime, bool noTakeLiq, bool speedUp, double monetaryValue){
    baseOrder.algoStrategy = "Vwap";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("maxPctVol", std::to_string(maxPctVol)));
    TagValueSPtr tag2(new TagValue("startTime", startTime));
    TagValueSPtr tag3(new TagValue("endTime", endTime));
    TagValueSPtr tag4(new TagValue("allowPastEndTime", allowPastEndTime ? "1" : "0"));
    TagValueSPtr tag5(new TagValue("noTakeLiq", noTakeLiq ? "1" : "0"));
    TagValueSPtr tag6(new TagValue("speedUp", speedUp ? "1" : "0"));
    TagValueSPtr tag7(new TagValue("monetaryValue", std::to_string(monetaryValue)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
    baseOrder.algoParams->push_back(tag6);
    baseOrder.algoParams->push_back(tag7);
}
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillVwapParams(baseOrder, 0.2,
                                           "09:00:00 CET", "16:00:00 CET", True, True, 100000)
        self.placeOrder(self.nextOrderId(), ContractSamples.USStockAtSmart(), baseOrder)
```

...

```
    @staticmethod
    def FillVwapParams(baseOrder: Order, maxPctVol: float, startTime: str,
                       endTime: str, allowPastEndTime: bool, noTakeLiq: bool,
                       monetaryValue: float):
        baseOrder.algoStrategy = "Vwap"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("maxPctVol", maxPctVol))
        baseOrder.algoParams.append(TagValue("startTime", startTime))
        baseOrder.algoParams.append(TagValue("endTime", endTime))
        baseOrder.algoParams.append(TagValue("allowPastEndTime",
                                             int(allowPastEndTime)))
        baseOrder.algoParams.append(TagValue("noTakeLiq", int(noTakeLiq)))
        baseOrder.algoParams.append(TagValue("monetaryValue", monetaryValue))
```

10.2.9.0.0.12 Balance Impact Risk

The `Balance Impact Risk` balances the market impact of trading the option with the risk of price change over the time horizon of the order. This strategy considers the user-assigned level of risk aversion to define the pace of the execution, along with the user-defined target percent of volume.

Parameter	Description	Values
maxPctVol	Maximum percentage of average daily volume	0.1 (10%) - 0.5 (50%)
riskAversion	Urgency/risk aversion	Get Done, Aggressive, Neutral, Passive
forceCompletion	Attempt completion by the end of the day	1 (true) or 0 (false)

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillBalanceImpactRiskParams(baseOrder, 0.1, "Aggressive", true);
            client.placeOrder(nextOrderId++, ContractSamples.USOptionContract(), baseOrder);
```

...

```
        public static void FillBalanceImpactRiskParams(Order baseOrder, double maxPctVol, string
      riskAversion, bool forceCompletion)
        {
            baseOrder.AlgoStrategy = "BalanceImpactRisk";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("maxPctVol", maxPctVol.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("riskAversion", riskAversion));
            baseOrder.AlgoParams.Add(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillBalanceImpactRiskParams(baseOrder, 0.1, "Aggressive", true);
        client.placeOrder(nextOrderId++, ContractSamples.USOptionContract(), baseOrder);
```

...

```
    public static void FillBalanceImpactRiskParams(Order baseOrder, double maxPctVol, String riskAversion,
      boolean forceCompletion) {

        baseOrder.algoStrategy("BalanceImpactRisk");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("maxPctVol", String.valueOf(maxPctVol)));
        baseOrder.algoParams().add(new TagValue("riskAversion", riskAversion));
        baseOrder.algoParams().add(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));

    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillBalanceImpactRiskParams(baseOrder, 0.10000000000000001, "Aggressive", True)
        client.placeOrder(increment(nextOrderId), ContractSamples.USOptionContract(), baseOrder)
```

...

```
        Public Shared Sub FillBalanceImpactRiskParams(baseOrder As Order, maxPctVol As Double, riskAversion
     As String, forceCompletion As Boolean)
            baseOrder.AlgoStrategy = "BalanceImpactRisk"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("maxPctVol", maxPctVol.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("riskAversion", riskAversion))
            baseOrder.AlgoParams.Add(New TagValue("forceCompletion", BooleantoString(forceCompletion)))
        End Sub
```

-
```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
```

...

```
    AvailableAlgoParams::FillBalanceImpactRiskParams(baseOrder, 0.1, "Aggressive", true);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
    void AvailableAlgoParams::FillBalanceImpactRiskParams(Order& baseOrder, double maxPctVol, std::string
      riskAversion, bool forceCompletion){
        baseOrder.algoStrategy = "BalanceImpactRisk";
        baseOrder.algoParams.reset(new TagValueList());
        TagValueSPtr tag1(new TagValue("maxPctVol", std::to_string(maxPctVol)));
        TagValueSPtr tag2(new TagValue("riskAversion", riskAversion));
        TagValueSPtr tag3(new TagValue("forceCompletion", forceCompletion ? "1" : "0"));
        baseOrder.algoParams->push_back(tag1);
        baseOrder.algoParams->push_back(tag2);
        baseOrder.algoParams->push_back(tag3);
    }
```

-
```
        baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
        AvailableAlgoParams.FillBalanceImpactRiskParams(baseOrder, 0.1,
                                                        "Aggressive", True)
        self.placeOrder(self.nextOrderId(), ContractSamples.USOptionContract(), baseOrder)
```

...

```
    @staticmethod
    def FillBalanceImpactRiskParams(baseOrder: Order, maxPctVol: float,
                                    riskAversion: str, forceCompletion: bool):
        baseOrder.algoStrategy = "BalanceImpactRisk"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("maxPctVol", maxPctVol))
        baseOrder.algoParams.append(TagValue("riskAversion", riskAversion))
        baseOrder.algoParams.append(TagValue("forceCompletion",
                                             int(forceCompletion)))
```

10.2.9.0.0.13 Minimise Impact

The `Minimise Impact` algo minimises market impact by slicing the order over time to achieve a market average without going over the given maximum percentage value.

Parameter	Description	Values
maxPctVol	Maximum percentage of average daily volume	0.1 (10%) - 0.5 (50%)

-
```
            Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
            AvailableAlgoParams.FillMinImpactParams(baseOrder, 0.3);
            client.placeOrder(nextOrderId++, ContractSamples.USOptionContract(), baseOrder);
```

...

```
        public static void FillMinImpactParams(Order baseOrder, double maxPctVol)
        {
            baseOrder.AlgoStrategy = "MinImpact";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("maxPctVol", maxPctVol.ToString()));
        }
```

-
```
        Order baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1);
```

...

```
        AvailableAlgoParams.FillMinImpactParams(baseOrder, 0.3);
        client.placeOrder(nextOrderId++, ContractSamples.USOptionContract(), baseOrder);
```

...

```
    public static void FillMinImpactParams(Order baseOrder, double maxPctVol) {

        baseOrder.algoStrategy("BalanceImpactRisk");
        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("maxPctVol", String.valueOf(maxPctVol)));

    }
```

-
```
        Dim baseOrder As Order = OrderSamples.LimitOrder("BUY", 1000, 1)
```

...

```
            AvailableAlgoParams.FillMinImpactParams(baseOrder, 0.29999999999999999)
            client.placeOrder(increment(nextOrderId), ContractSamples.USOptionContract(), baseOrder)
```

...

```
            Public Shared Sub FillMinImpactParams(baseOrder As Order, maxPctVol As Double)

                baseOrder.AlgoStrategy = "MinImpact"
                baseOrder.AlgoParams = New List(Of TagValue)
                baseOrder.AlgoParams.Add(New TagValue("maxPctVol", maxPctVol.ToString()))
            End Sub
```

- ```
    Order baseOrder = OrderSamples::LimitOrder("BUY", 1000, 1);
  ```

...

```
    AvailableAlgoParams::FillMinImpactParams(baseOrder, 0.3);
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStockAtSmart(), baseOrder);
```

...

```
void AvailableAlgoParams::FillMinImpactParams(Order& baseOrder, double maxPctVol){
    baseOrder.algoStrategy = "MinImpact";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("maxPctVol", std::to_string(maxPctVol)));
    baseOrder.algoParams->push_back(tag1);
}
```

- ```
          baseOrder = OrderSamples.LimitOrder("BUY", 1000, 1)
  ```

...

```
            AvailableAlgoParams.FillMinImpactParams(baseOrder, 0.3)
            self.placeOrder(self.nextOrderId(), ContractSamples.USOptionContract(), baseOrder)
```

...

```
    @staticmethod
    def FillMinImpactParams(baseOrder: Order, maxPctVol: float):
        baseOrder.algoStrategy = "MinImpact"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("maxPctVol", maxPctVol))
```

10.2.10 CSFB Algorithms

10.2.10.0.0.1 CSFB Algorithm Parameters

The socket-based APIs support the following CSFB Algo Strategies: CSFB Algos are not available in paper accounts.

Inline

Pathfinder

Blast

Guerrilla

Sniper

Crossfinder

I Would

Float Guerrilla

VWAP

Volume Inline

TWAP

Pre/Post

Close

Reserve

Float

Tex

LightPool

10b-18

Auction Algo

The following table lists all available CSFB algo strategies and parameters supported by the API.

10.2.10.0.0.2 Inline

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
MinPercent	Minimum percentage of volume	Integer	Range: 0 - 99
MaxPercent	Maximum percentage volume	Integer	Range: 0 - 99
DisplaySize	Size display for iceberg/reserve	Integer	
Auction		String	"Default", "Include_All", "Include_Open_Only", "Include_Close_Only", "Exclude_All", "Imbalance_Only"
Blockfinder	Enables block finding	Boolean	"1", "0" (for Java "true", "false")
BlockPrice	Price of the block. Cannot violate price of the algo order.	Double	
MinBlockSize	Minimum block quantity	Integer	
MaxBlockSize	Maximum block quantity	Integer	
IWouldPrice		Double	

Example CSFB Inline Algo

-
```
Contract contract = new Contract();
```

```
            contract.Symbol = "IBKR";
            contract.SecType = "STK";
            contract.Exchange = "CSFBALGO"; // must be direct-routed to CSFBALGO
            contract.Currency = "USD"; // only available for US stocks
```

...

```
            AvailableAlgoParams.FillCSFBInlineParams(baseOrder, "10:00:00 EST", "16:00:00 EST", "Patient",
      10, 20, 100, "Default", false, 40, 100, 100, 35);
            client.placeOrder(nextOrderId++, ContractSamples.CSFBContract(), baseOrder);
```

...

```
        public static void FillCSFBInlineParams(Order baseOrder, string startTime, string endTime, string
      execStyle, int minPercent,
        int maxPercent, int displaySize, string auction, bool blockFinder, double blockPrice, int
      minBlockSize, int maxBlockSize, double iWouldPrice)
        {

            // must be direct-routed to "CSFBALGO"

            baseOrder.AlgoStrategy = "INLINE";
            baseOrder.AlgoParams = new List<TagValue>();
            baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
            baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
            baseOrder.AlgoParams.Add(new TagValue("execStyle", execStyle));
            baseOrder.AlgoParams.Add(new TagValue("minPercent", minPercent.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("maxPercent", maxPercent.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("displaySize", displaySize.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("auction", auction));
            baseOrder.AlgoParams.Add(new TagValue("blockFinder", blockFinder ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("blockPrice", blockPrice.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("minBlockSize", minBlockSize.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("maxBlockSize", maxBlockSize.ToString()));
            baseOrder.AlgoParams.Add(new TagValue("iWouldPrice", iWouldPrice.ToString()));
        }
```

-
```
        Contract contract = new Contract();
        contract.symbol("IBKR");
        contract.secType("STK");
        contract.exchange("CSFBALGO"); // must be direct-routed to CSFBALGO
        contract.currency("USD");    // only available for US stocks
```

...

```
        AvailableAlgoParams.FillCSFBInlineParams(baseOrder, "10:00:00 EST", "16:00:00 EST", "Patient", 10,
      20, 100, "Default", false, 40, 100, 100, 35 );
        client.placeOrder(nextOrderId++, ContractSamples.CSFBContract(), baseOrder);
```

...

```
    public static void FillCSFBInlineParams(Order baseOrder, String startTime, String endTime, String
      execStyle, int minPercent,
                                            int maxPercent, int displaySize, String auction, boolean
      blockFinder, double blockPrice,
                                            int minBlockSize, int maxBlockSize, double iWouldPrice) {

        // must be direct-routed to "CSFBALGO"

        baseOrder.algoStrategy("INLINE");

        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("StartTime", startTime));
        baseOrder.algoParams().add(new TagValue("EndTime", endTime));
        baseOrder.algoParams().add(new TagValue("ExecStyle", execStyle));
        baseOrder.algoParams().add(new TagValue("MinPercent", String.valueOf(minPercent)));
        baseOrder.algoParams().add(new TagValue("MaxPercent", String.valueOf(maxPercent)));
        baseOrder.algoParams().add(new TagValue("DisplaySize", String.valueOf(displaySize)));
        baseOrder.algoParams().add(new TagValue("Auction", auction));
        baseOrder.algoParams().add(new TagValue("BlockFinder", blockFinder ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("BlockPrice", String.valueOf(blockPrice)));
        baseOrder.algoParams().add(new TagValue("MinBlockSize", String.valueOf(minBlockSize)));
        baseOrder.algoParams().add(new TagValue("MaxBlockSize", String.valueOf(maxBlockSize)));
        baseOrder.algoParams().add(new TagValue("IWouldPrice", String.valueOf(iWouldPrice)));

    }
```

-
```
            Dim contract As Contract = New Contract()
            contract.Symbol = "IBKR"
            contract.SecType = "STK"
            contract.Exchange = "CSFBALGO" 'must be direct-routed to CSFBALGO
            contract.Currency = "USD" 'only available for US stocks
```

...

```
        AvailableAlgoParams.FillCSFBInlineParams(baseOrder, "10:00:00 EST", "16:00:00 EST", "Patient", 10,
       20, 100, "Default", False, 40, 100, 100, 35)
        client.placeOrder(increment(nextOrderId), ContractSamples.CSFBContract(), baseOrder)
```

...

```
        Public Shared Sub FillCSFBInlineParams(baseOrder As Order, startTime As String, endTime As String,
       execStyle As String, minPercent As Integer, maxPercent As Integer, displaySize As Integer, auction As
       String,
            blockFinder As Boolean, blockPrice As Double, minBlockSize As Integer, maxBlockSize As Integer,
       iWouldPrice As Double)

            'Must be direct-routed to "CSFBALGO"

            baseOrder.AlgoStrategy = "INLINE"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("execStyle", execStyle))
            baseOrder.AlgoParams.Add(New TagValue("minPercent", minPercent.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("maxPercent", maxPercent.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("displaySize", displaySize.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("auction", auction))
            baseOrder.AlgoParams.Add(New TagValue("blockFinder", BooleantoString(blockFinder)))
            baseOrder.AlgoParams.Add(New TagValue("blockPrice", blockPrice.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("minBlockSize", minBlockSize.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("maxBlockSize", maxBlockSize.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("iWouldPrice", iWouldPrice.ToString()))
        End Sub
```

-
```
    Contract contract;
    contract.symbol = "IBKR";
    contract.secType = "STK";
    contract.exchange = "CSFBALGO";
    contract.currency = "USD";
```

...

```
    AvailableAlgoParams::FillCSFBInlineParams(baseOrder, "10:00:00 EST", "16:00:00 EST", "Patient", 10, 20,
       100, "Default", false, 40, 100, 100, 35);
    m_pClient->placeOrder(m_orderId++, ContractSamples::CSFBContract(), baseOrder);
```

...

```
void AvailableAlgoParams::FillCSFBInlineParams(Order baseOrder, std::string startTime, std::string endTime,
       std::string execStyle, int minPercent,
        int maxPercent, int displaySize, std::string auction, bool blockFinder, double blockPrice, int
      minBlockSize, int maxBlockSize, double iWouldPrice){

    // must be direct-routed to "CSFBALGO"

    baseOrder.algoStrategy = "INLINE";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("StartTime", startTime));
    TagValueSPtr tag2(new TagValue("EndTime", endTime));
    TagValueSPtr tag3(new TagValue("ExecStyle", execStyle));
    TagValueSPtr tag4(new TagValue("MinPercent", std::to_string(minPercent)));
    TagValueSPtr tag5(new TagValue("MaxPercent", std::to_string(maxPercent)));
    TagValueSPtr tag6(new TagValue("DisplaySize", std::to_string(displaySize)));
    TagValueSPtr tag7(new TagValue("Auction", auction));
    TagValueSPtr tag8(new TagValue("BlockFinder", blockFinder ? "1" : "0"));
    TagValueSPtr tag9(new TagValue("BlockPrice", std::to_string(blockPrice)));
    TagValueSPtr tag10(new TagValue("MinBlockSize", std::to_string(minBlockSize)));
    TagValueSPtr tag11(new TagValue("MaxBlockSize", std::to_string(maxBlockSize)));
    TagValueSPtr tag12(new TagValue("IWouldPrice", std::to_string(iWouldPrice)));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
    baseOrder.algoParams->push_back(tag6);
    baseOrder.algoParams->push_back(tag7);
    baseOrder.algoParams->push_back(tag8);
    baseOrder.algoParams->push_back(tag9);
    baseOrder.algoParams->push_back(tag10);
    baseOrder.algoParams->push_back(tag11);
    baseOrder.algoParams->push_back(tag12);
}
```

-
```
        contract = Contract()
        contract.symbol = "IBKR"
        contract.secType = "STK"
```

```
        contract.exchange = "CSFBALGO"
        contract.currency = "USD"

...

        AvailableAlgoParams.FillCSFBInlineParams(baseOrder,
                                                 "10:00:00 EST", "16:00:00 EST", "Patient",
                                                 10, 20, 100, "Default", False, 40, 100, 100, 35)

        self.placeOrder(self.nextOrderId(), ContractSamples.CSFBContract(), baseOrder)


...

    @staticmethod
    def FillCSFBInlineParams(baseOrder: Order, startTime: str,
                             endTime: str, execStyle: str,
                             minPercent: int, maxPercent: int,
                             displaySize: int, auction: str,
                             blockFinder: bool, blockPrice: float,
                             minBlockSize: int, maxBlockSize: int, iWouldPrice: float):
        # must be direct-routed to "CSFBALGO"
        baseOrder.algoStrategy = "INLINE"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("StartTime", startTime))
        baseOrder.algoParams.append(TagValue("EndTime", endTime))
        baseOrder.algoParams.append(TagValue("ExecStyle", execStyle))
        baseOrder.algoParams.append(TagValue("MinPercent", minPercent))
        baseOrder.algoParams.append(TagValue("MaxPercent", maxPercent))
        baseOrder.algoParams.append(TagValue("DisplaySize", displaySize))
        baseOrder.algoParams.append(TagValue("Auction", auction))
        baseOrder.algoParams.append(TagValue("BlockFinder", int(blockFinder)))
        baseOrder.algoParams.append(TagValue("BlockPrice", blockPrice))
        baseOrder.algoParams.append(TagValue("MinBlockSize", minBlockSize))
        baseOrder.algoParams.append(TagValue("MaxBlockSize", maxBlockSize))
        baseOrder.algoParams.append(TagValue("IWouldPrice", iWouldPrice))
```

10.2.10.0.0.3 Pathfinder

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
DisplaySize	Size display for iceberg/reserve	Integer	

10.2.10.0.0.4 Blast

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
DisplaySize	Size display for iceberg/reserve	Integer	

10.2.10.0.0.5 Guerrilla

Parameter

Description

Type

Syntax/Values

StartTime

Start time

Time

9:00:00 EST

EndTime

End time

Time

15:00:00 EST

ExecStyle

Execution Style

String

"Patient", "Normal", "Aggressive"

MinPercent

MaxPercent

Maximum percentage volume

Integer

range: 0 - 99

DisplaySize

Size display for iceberg/reserve

Integer

Blockfinder

Enables block finding

Boolean

"1", "0" (for Java "true", "false")

BlockPrice

Price of the block. Cannot violate price of the algo order.

Double

MinBlockSize

Minimum block quantity

Integer

MaxBlockSize

Maximum block quantity

Integer

IWouldPrice

Double

10.2.10.0.0.6 Sniper

Parameter

Description

Type

Syntax/Values

StartTime

Start time

Time

9:00:00 EST

EndTime

End time

Time

15:00:00 EST

ExecStyle

Execution Style

String

"Patient", "Normal", "Aggressive"

MaxPercent

Maximum percentage volume

Integer

range: 0 - 99

DisplaySize

Size display for iceberg/reserve

Integer

Blockfinder

Enables block finding

Boolean

"1", "0" (for Java "true", "false")

BlockPrice

Price of the block. Cannot violate price of the algo order.

Double

MinBlockSize

Minimum block quantity

Integer

MaxBlockSize

Maximum block quantity

Integer

10.2.10.0.0.7 Crossfinder

Parameter

Description

Type

Syntax/Values

StartTime

Start time

Time

9:00:00 EST

EndTime

End time

Time

15:00:00 EST

ExecStyle

Execution Style

String

"Patient", "Normal", "Aggressive"

MaxPercent

Maximum percentage volume

Integer

range: 0 - 99

DisplaySize

Size display for iceberg/reserve

Integer

Blockfinder

Enables block finding

Boolean

"1", "0" (for Java "true", "false")

BlockPrice

Price of the block. Cannot violate price of the algo order.

Double

MinBlockSize

Minimum block quantity

Integer

MaxBlockSize

Maximum block quantity

Integer

10.2.10.0.0.8 I Would

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
MinPercent			
MaxPercent	Maximum percentage volume	Integer	range: 0 - 99
DisplaySize	Size display for iceberg/reserve	Integer	
Blockfinder	Enables block finding	Boolean	"1", "0" (for Java "true", "false")
BlockPrice	Price of the block. Cannot violate price of the algo order.	Double	
MinBlockSize	Minimum block quantity	Integer	
MaxBlockSize	Maximum block quantity	Integer	

10.2.10.0.0.9 Float Guerrilla

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
MinPercent			
MaxPercent	Maximum percentage volume	Integer	range: 0 - 99
DisplaySize	Size display for iceberg/reserve	Integer	
Blockfinder	Enables block finding	Boolean	"1", "0" (for Java "true", "false")
BlockPrice	Price of the block. Cannot violate price of the algo order.	Double	
MinBlockSize	Minimum block quantity	Integer	
MaxBlockSize	Maximum block quantity	Integer	
IWouldPrice		Double	

10.2.10.0.0.10 VWAP

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
MaxPercent	Maximum percentage volume	Integer	range: 0 - 99
Auction		String	"Default", "Include_All", "Include_↩ Open_Only", "Include_Close_Only", "Exclude_All", "Imbalance_Only"
Blockfinder	Enables block finding	Boolean	"1", "0" (for Java "true", "false")
BlockPrice	Price of the block. Cannot violate price of the algo order.	Double	
MinBlockSize	Minimum block quantity	Integer	
MaxBlockSize	Maximum block quantity	Integer	
IWouldPrice		Double	

10.2.10.0.0.11 Volume Inline

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
MinPercent			
MaxPercent	Maximum percentage volume	Integer	range: 0 - 99
DisplaySize	Size display for iceberg/reserve	Integer	
Auction		String	"Default", "Include_All", "Include_↩ Open_Only", "Include_Close_Only", "Exclude_All", "Imbalance_Only"
Blockfinder	Enables block finding	Boolean	"1", "0" (for Java "true", "false")
BlockPrice	Price of the block. Cannot violate price of the algo order.	Double	
MinBlockSize	Minimum block quantity	Integer	
MaxBlockSize	Maximum block quantity	Integer	
IWouldPrice		Double	

10.2.10.0.0.12 TWAP

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
MaxPercent	Maximum percentage volume	Integer	range: 0 - 99
Auction		String	"Default", "Include_All", "Include_↩ Open_Only", "Include_Close_Only", "Exclude_All", "Imbalance_Only"
Blockfinder	Enables block finding	Boolean	"1", "0" (for Java "true", "false")
BlockPrice	Price of the block. Cannot violate price of the algo order.	Double	
MinBlockSize	Minimum block quantity	Integer	
MaxBlockSize	Maximum block quantity	Integer	
IWouldPrice		Double	

10.2.10.0.0.13 Pre/Post

Parameter

Description

Type

Syntax/Values

StartTime

Start time

Time

9:00:00 EST

EndTime

End time

Time

15:00:00 EST

DisplaySize

Size display for iceberg/reserve

Integer

10.2.10.0.0.14 Close

Parameter

Description

Type

Syntax/Values

StartTime

Start time

Time

9:00:00 EST

EndTime

End time

Time

15:00:00 EST

10.2.10.0.0.15 Reserve

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
MaxPercent	Maximum percentage volume	Integer	range: 0 - 99
DisplaySize	Size display for iceberg/reserve	Integer	
BlockFinder	Enables block finding	Boolean	"1", "0" (for Java "true", "false")
BlockPrice	Price of the block. Cannot violate price of the algo order.	Double	
MinBlockSize	Minimum block quantity	Integer	

10.2.10.0.0.16 Float

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
MaxPercent	Maximum percentage volume	Integer	range: 0 - 99
DisplaySize	Size display for iceberg/reserve	Integer	
Blockfinder	Enables block finding	Boolean	"1", "0" (for Java "true", "false")
BlockPrice	Price of the block. Cannot violate price of the algo order.	Double	
MinBlockSize	Minimum block quantity	Integer	
MaxBlockSize	Maximum block quantity	Integer	

10.2.10.0.0.17 Tex

Parameter

Description

Type

Syntax/Values

StartTime

Start time

Time

9:00:00 EST

EndTime

End time

Time

15:00:00 EST

ExecStyle

Execution Style

String

"Patient", "Normal", "Aggressive"

MinPercent

Minimum percentage of volume

Integer

Range: 0 - 99

MaxPercent

Maximum percentage volume

Integer

Range: 0 - 99

DisplaySize

Size display for iceberg/reserve

Integer

Auction

String

"Default", "Include_All", "Include_Open_Only", "Include_Close_Only", "Exclude_All", "Imbalance_Only"

Blockfinder

Enables block finding

Boolean

"1", "0" (for Java "true", "false")

BlockPrice

Price of the block. Cannot violate price of the algo order.

Double

MinBlockSize

Minimum block quantity

Integer

MaxBlockSize

Maximum block quantity

Integer

IWouldPrice

Double

10.2.10.0.0.18 LightPool

Parameter

Description

Type

Syntax/Values

StartTime

Start time

Time

9:00:00 EST

EndTime

End time

Time

15:00:00 EST

ExecStyle

Execution Style

String

"Patient", "Normal", "Aggressive"

DisplaySize

Size display for iceberg/reserve

Integer

10.2.10.0.0.19 10b-18

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
MaxPercent	Maximum percentage volume	Integer	Range: 0 - 99

10.2.10.0.0.20 Auction Algo

Parameter	Description	Type	Syntax/Values
ExecStyle	Execution Style	String	"Patient", "Normal", "Aggressive"
DisplaySize	Size display for iceberg/reserve	Integer	

10.2.11 Jefferies Algorithms

10.2.11.0.0.1 Jefferies Algorithm

The Jefferies Algos are available with the socket-based API languages (Java, C#/.NET, Python, C++).

- It is recommended to first try to `create the Jefferies algo in TWS` to see the most current available field values.
- Jefferies algos are only available in live accounts.
- Some fields have default values and are optional in TWS but must be explicitly specified in the API.

10.2.11.0.0.2 Available Jefferies Algos

VWAP

TWAP

VolPart

Blitz

Strike

Seek

Darkseek

Post

MultiScale

Opener

Finale

Portfolio

Patience

Pairs - Ratio

Pairs - Net Return

Pairs - Arb

Trader

The following table lists all available Jefferies algo strategies and parameters supported by the

API.

10.2.11.0.0.3 VWAP (VWAP)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
Relative Limit		Double	Positive and negative values allowed.
MaxVolumeRate		Double	
ExcludeAuctions	Define auction participation	String	"Exclude_Both", "Include_Open", "Include_Close", "Include_Both"
TriggerPrice		Double	Any positive value, no max.
WoWPrice	Would or Work - The price at which the user is willing to complete the order. Used if no WoW reference is specified.	Double	
WowReference	Used with WoW price field. If WoW price is not submitted, the Reference price can be submitted for processing.	String	"Market", "Inside_NBBO_Price", "Arrival Price", "PNC", "Open", "BPS_Arrival", "Price", "OPP", "Midpoint"
MinFillSize	Minimum number of share per execution for non-displayed liquidity. Rounded down to closest lot size.	Integer	
WoWOrderPct	Max percent of the order on which WoW can work.	Integer	
WoWMode		String	"BLITZ", "DARKSeek", "Seek_Passive", "Seek_Active", "Seek_Aggressive", "Volume_10%", "Volume_20%", "Volume_30%","VWAP_Day", "Patience"
IsBuyBack	The algo should engage in SEC Rule 10b-18 restrictions for buy back in US securities.	String	"Yes", "No"

Example Jefferies VWAP Algo

-
```
Contract contract = new Contract();
contract.Symbol = "AAPL";
contract.SecType = "STK";
contract.Exchange = "JEFFALGO"; // must be direct-routed to JEFFALGO
contract.Currency = "USD"; // only available for US stocks
```

...

```
AvailableAlgoParams.FillJefferiesVWAPParams(baseOrder, "10:00:00 EST", "16:00:00 EST", 10, 10,
"Exclude_Both", 130, 135, 1, 10, "Patience", false, "Midpoint");
client.placeOrder(nextOrderId++, ContractSamples.JefferiesContract(), baseOrder);
```

...

```
public static void FillJefferiesVWAPParams(Order baseOrder, string startTime, string endTime,
double relativeLimit,
  double maxVolumeRate, string excludeAuctions, double triggerPrice, double wowPrice, int minFillSize
, double wowOrderPct,
  string wowMode, bool isBuyBack, string wowReference)
  {
      baseOrder.AlgoStrategy = "VWAP";
      baseOrder.AlgoParams = new List<TagValue>();
      baseOrder.AlgoParams.Add(new TagValue("startTime", startTime));
      baseOrder.AlgoParams.Add(new TagValue("endTime", endTime));
      baseOrder.AlgoParams.Add(new TagValue("relativeLimit", relativeLimit.ToString()));
      baseOrder.AlgoParams.Add(new TagValue("maxVolumeRate", maxVolumeRate.ToString()));
      baseOrder.AlgoParams.Add(new TagValue("excludeAuctions", excludeAuctions));
      baseOrder.AlgoParams.Add(new TagValue("triggerPrice", triggerPrice.ToString()));
      baseOrder.AlgoParams.Add(new TagValue("wowPrice", wowPrice.ToString()));
      baseOrder.AlgoParams.Add(new TagValue("minFillSize", minFillSize.ToString()));
      baseOrder.AlgoParams.Add(new TagValue("wowOrderPct", wowOrderPct.ToString()));
```

```
            baseOrder.AlgoParams.Add(new TagValue("wowMode", wowMode));
            baseOrder.AlgoParams.Add(new TagValue("IsBuyBack", isBuyBack ? "1" : "0"));
            baseOrder.AlgoParams.Add(new TagValue("wowReference", wowReference));
        }
```

-
```
        Contract contract = new Contract();
        contract.symbol("AAPL");
        contract.secType("STK");
        contract.exchange("JEFFALGO"); // must be direct-routed to JEFFALGO
        contract.currency("USD");    // only available for US stocks
```

...

```
        AvailableAlgoParams.FillJefferiesVWAPParams(baseOrder, "10:00:00 EST", "16:00:00 EST", 10, 10, "
    Exclude_Both", 130, 135, 1, 10, "Patience", false, "Midpoint");
        client.placeOrder(nextOrderId++, ContractSamples.JefferiesContract(), baseOrder);
```

...

```
        public static void FillJefferiesVWAPParams(Order baseOrder, String startTime, String endTime,
    double relativeLimit,
            double maxVolumeRate, String excludeAuctions, double triggerPrice, double wowPrice, int
    minFillSize, double wowOrderPct,
            String wowMode, boolean isBuyBack, String wowReference) {

        // must be direct-routed to "JEFFALGO"

        baseOrder.algoStrategy("VWAP");

        baseOrder.algoParams(new ArrayList<>());
        baseOrder.algoParams().add(new TagValue("StartTime", startTime));
        baseOrder.algoParams().add(new TagValue("EndTime", endTime));
        baseOrder.algoParams().add(new TagValue("RelativeLimit", String.valueOf(relativeLimit)));
        baseOrder.algoParams().add(new TagValue("MaxVolumeRate", String.valueOf(maxVolumeRate)));
        baseOrder.algoParams().add(new TagValue("ExcludeAuctions", excludeAuctions));
        baseOrder.algoParams().add(new TagValue("TriggerPrice", String.valueOf(triggerPrice)));
        baseOrder.algoParams().add(new TagValue("WowPrice", String.valueOf(wowPrice)));
        baseOrder.algoParams().add(new TagValue("MinFillSize", String.valueOf(minFillSize)));
        baseOrder.algoParams().add(new TagValue("WowOrderPct", String.valueOf(wowOrderPct)));
        baseOrder.algoParams().add(new TagValue("WowMode", wowMode));
        baseOrder.algoParams().add(new TagValue("IsBuyBack", isBuyBack ? "1" : "0"));
        baseOrder.algoParams().add(new TagValue("WowReference", wowReference));

    }
```

-
```
            Dim contract As Contract = New Contract()
            contract.Symbol = "AAPL"
            contract.SecType = "STK"
            contract.Exchange = "JEFFALGO" 'must me direct-routed to JEFFALGO
            contract.Currency = "USD" 'only available for US stocks
```

...

```
        AvailableAlgoParams.FillJefferiesVWAPParams(baseOrder, "10:00:00 EST", "16:00:00 EST", 10, 10,
    "Exclude_Both", 130, 135, 1, 10, "Patience", False, "Midpoint")
        client.placeOrder(increment(nextOrderId), ContractSamples.JefferiesContract(), baseOrder)
```

...

```
        Public Shared Sub FillJefferiesVWAPParams(baseOrder As Order, startTime As String, endTime As
    String, relativeLimit As Double, maxVolumeRate As Double, excludeAuctions As String,
            triggerPrice As Double, wowPrice As Double, minFillSize As Integer, wowOrderPct As Double,
    wowMode As String, isBuyBack As Boolean, wowReference As String)

            'Must be direct-routed to "JEFFALGO"

            baseOrder.AlgoStrategy = "VWAP"
            baseOrder.AlgoParams = New List(Of TagValue)
            baseOrder.AlgoParams.Add(New TagValue("startTime", startTime))
            baseOrder.AlgoParams.Add(New TagValue("endTime", endTime))
            baseOrder.AlgoParams.Add(New TagValue("relativeLimit", relativeLimit.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("maxVolumeRate", maxVolumeRate.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("excludeAuctions", excludeAuctions))
            baseOrder.AlgoParams.Add(New TagValue("triggerPrice", triggerPrice.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("wowPrice", wowPrice.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("minFillSize", minFillSize.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("wowOrderPct", wowOrderPct.ToString()))
            baseOrder.AlgoParams.Add(New TagValue("wowMode", wowMode))
            baseOrder.AlgoParams.Add(New TagValue("isBuyBack", BooleantoString(isBuyBack)))
            baseOrder.AlgoParams.Add(New TagValue("wowReference", wowReference))
        End Sub
```

- ```
    Contract contract;
    contract.symbol = "AAPL";
    contract.secType = "STK";
    contract.exchange = "JEFFALGO"; // must be direct-routed to JEFALGO
    contract.currency = "USD"; // only available for US stocks
  ```

...

```
    AvailableAlgoParams::FillJefferiesVWAPParams(baseOrder, "10:00:00 EST", "16:00:00 EST", 10, 10, "
      Exclude_Both", 130, 135, 1, 10, "Patience", false, "Midpoint");
    m_pClient->placeOrder(m_orderId++, ContractSamples::JefferiesContract(), baseOrder);
```

...

```
void AvailableAlgoParams::FillJefferiesVWAPParams(Order baseOrder, std::string startTime, std::string
      endTime, double relativeLimit,
        double maxVolumeRate, std::string excludeAuctions, double triggerPrice, double wowPrice, int
      minFillSize, double wowOrderPct,
        std::string wowMode, bool isBuyBack, std::string wowReference){
    baseOrder.algoStrategy = "VWAP";
    baseOrder.algoParams.reset(new TagValueList());
    TagValueSPtr tag1(new TagValue("StartTime", startTime));
    TagValueSPtr tag2(new TagValue("EndTime", endTime));
    TagValueSPtr tag3(new TagValue("RelativeLimit", std::to_string(relativeLimit)));
    TagValueSPtr tag4(new TagValue("MaxVolumeRate", std::to_string(maxVolumeRate)));
    TagValueSPtr tag5(new TagValue("ExcludeAuctions", excludeAuctions));
    TagValueSPtr tag6(new TagValue("TriggerPrice", std::to_string(triggerPrice)));
    TagValueSPtr tag7(new TagValue("WowPrice", std::to_string(wowPrice)));
    TagValueSPtr tag8(new TagValue("MinFillSize", std::to_string(minFillSize)));
    TagValueSPtr tag9(new TagValue("WowOrderPct", std::to_string(wowOrderPct)));
    TagValueSPtr tag10(new TagValue("WowMode", wowMode));
    TagValueSPtr tag11(new TagValue("IsBuyBack", isBuyBack ? "1" : "0"));
    TagValueSPtr tag12(new TagValue("WowReference", wowReference));
    baseOrder.algoParams->push_back(tag1);
    baseOrder.algoParams->push_back(tag2);
    baseOrder.algoParams->push_back(tag3);
    baseOrder.algoParams->push_back(tag4);
    baseOrder.algoParams->push_back(tag5);
    baseOrder.algoParams->push_back(tag6);
    baseOrder.algoParams->push_back(tag7);
    baseOrder.algoParams->push_back(tag8);
    baseOrder.algoParams->push_back(tag9);
    baseOrder.algoParams->push_back(tag10);
    baseOrder.algoParams->push_back(tag11);
    baseOrder.algoParams->push_back(tag12);
}
```

- ```
        contract = Contract()
        contract.symbol = "AAPL"
        contract.secType = "STK"
        contract.exchange = "JEFFALGO"
        contract.currency = "USD"
  ```

...

```
        AvailableAlgoParams.FillJefferiesVWAPParams(baseOrder,
                                                    "10:00:00 EST", "16:00:00 EST", 10, 10, "Exclude_Both",
                                                    130, 135, 1, 10, "Patience", False, "Midpoint")

        self.placeOrder(self.nextOrderId(), ContractSamples.JefferiesContract(), baseOrder)
```

...

```
    @staticmethod
    def FillJefferiesVWAPParams(baseOrder: Order, startTime: str,
                                endTime: str, relativeLimit: float,
                                maxVolumeRate: float, excludeAuctions: str,
                                triggerPrice: float, wowPrice: float,
                                minFillSize: int, wowOrderPct: float,
                                wowMode: str, isBuyBack: bool, wowReference: str):
        # must be direct-routed to "JEFFALGO"
        baseOrder.algoStrategy = "VWAP"
        baseOrder.algoParams = []
        baseOrder.algoParams.append(TagValue("StartTime", startTime))
        baseOrder.algoParams.append(TagValue("EndTime", endTime))
        baseOrder.algoParams.append(TagValue("RelativeLimit", relativeLimit))
        baseOrder.algoParams.append(TagValue("MaxVolumeRate", maxVolumeRate))
        baseOrder.algoParams.append(TagValue("ExcludeAuctions", excludeAuctions))
        baseOrder.algoParams.append(TagValue("TriggerPrice", triggerPrice))
        baseOrder.algoParams.append(TagValue("WowPrice", wowPrice))
        baseOrder.algoParams.append(TagValue("MinFillSize", minFillSize))
        baseOrder.algoParams.append(TagValue("WowOrderPct", wowOrderPct))
        baseOrder.algoParams.append(TagValue("WowMode", wowMode))
        baseOrder.algoParams.append(TagValue("IsBuyBack", int(isBuyBack)))
        baseOrder.algoParams.append(TagValue("WowReference", wowReference))
```

10.2.11.0.0.4 TWAP (TWAP)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
Relative Limit		Double	Positive and negative values allowed.
MaxVolumeRate		Double	
ExcludeAuctions	Define auction participation	String	"Exclude_Both", "Include_Open", "Include_Close", "Include_Both"
TradingSession	Denotes the trading session for the order.	String	"DAY", "PRE-OPEN", "AFTER-HOURS"
WoWPrice	Would or Work - The price at which the user is willing to complete the order. Used if no WoW reference is specified.	Double	
WowReference	Used with WoW price field. If WoW price is not submitted, the Reference price can be submitted for processing.	String	"Market", "Inside_NBBO_Price", "Arrival Price", "PNC", "Open", "BPS_Arrival", "Price", "OPP", "Midpoint"
MinFillSize	Minimum number of share per execution. Rounded down to closest lot size.	Integer	
WoWOrderPct	Max percent of the order on which WoW can work.	Integer	
WoWMode		String	"BLITZ", "DARKSeek", "Seek_Passive", "Seek_Active", "Seek_Aggressive", "Volume_10%", "Volume_20%", "Volume_30%","VWAP_Day", "Patience"
IsBuyBack	The algo should engage in SEC Rule 10b-18 restrictions for buy back in US securities.	String	"Yes", "No"

10.2.11.0.0.5 VolPart (VOLPART)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
Relative Limit		Double	Positive and negative values allowed.
MaxVolumeRate	Volume limit	Double	
DarkVolumeRate	Dark volume limit	Double	
ExcludeAuctions	Define auction participation	String	"Exclude_Both", "Include_Open", "Include_Close", "Include_Both"
TradingSession	Denotes the trading session for the order.	String	"DAY", "PRE-OPEN", "AFTER-HOURS"
WoWPrice	Would or Work - The price at which the user is willing to complete the order. Used if no WoW reference is specified.	Double	
WowReference	Used with WoW price field. If WoW price is not submitted, the Reference price can be submitted for processing.	String	"Market", "Inside_NBBO_Price", "Arrival Price", "PNC", "Open", "BPS_Arrival", "Price", "OPP", "Midpoint"

Parameter	Description	Type	Syntax/Values
MinFillSize	Minimum number of share per execution for non-displayed liquidity. Rounded down to closest lot size.	Integer	
WoWOrderPct	Max percent of the order on which WoW can work.	Integer	
WoWMode		String	"BLITZ", "DARKSeek", "Seek_Passive", "Seek_Active", "Seek_Aggressive", "Volume_10%", "Volume_20%", "Volume_30%","VWAP_Day", "Patience"
IsBuyBack	The algo should engage in SEC Rule 10b-18 restrictions for buy back in US securities.	String	"Yes", "No"

10.2.11.0.0.6 Blitz (BLITZ)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
Relative Limit		Double	Positive and negative values allowed.
DisplaySize		Integer	
TradingStyle		String	"Price Improvement", "Opportunistic", "Get-It-Done", "No_Post"
IsBuyBack	The algo should engage in SEC Rule 10b-18 restrictions for buy back in US securities.	String	"Yes", "No"
TradingSession	Denotes the trading session for the order.	String	"DAY", "PRE-OPEN", "AFTER-HOURS"

10.2.11.0.0.7 Strike (STRIKE)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
Relative Limit		Double	Positive and negative values allowed.
MaxVolumeRate		Double	
Urgency		String	"Passive", "Active", "Aggressive"
ExcludeAuctions	Define auction participation	String	"Exclude_Both", "Include_Open", "Include_Close", "Include_Both"
WoWPrice	Would or Work - The price at which the user is willing to complete the order. Used if no WoW reference is specified.	Double	
WowReference	Used with WoW price field. If WoW price is not submitted, the Reference price can be submitted for processing.	String	"Market", "Inside_NBBO_Price", "Arrival Price", "PNC", "Open", "BPS_Arrival", "Price", "OPP", "Midpoint"

Parameter	Description	Type	Syntax/Values
MinFillSize	Minimum number of share per execution for non-displayed liquidity. Rounded down to closest lot size.	Integer	
WoWOrderPct	Max percent of the order on which WoW can work.	Integer	
WoWMode		String	"BLITZ", "DARKSeek", "Seek_↩ Passive", "Seek_Active", "Seek↩ _Aggressive", "Volume_10%", "↩ Volume_20%", "Volume_30%","VWA↩ P_Day", "Patience"

10.2.11.0.0.8 Seek (SEEK)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
Relative Limit		Double	Positive and negative values allowed.
ExcludeAuctions	Define auction participation	String	"Exclude_Both", "Include_Open", "↩ Include_Close", "Include_Both"
Urgency		String	"Passive_Minus", "Passive", "Active", "Active_Plus", "Aggressive"
MaxVolumeRate		Double	
WoWPrice	Would or Work - The price at which the user is willing to complete the order. Used if no WoW reference is specified.	Double	
WowReference	Used with WoW price field. If WoW price is not submitted, the Reference price can be submitted for processing.	String	"Market", "Inside_NBBO_Price", "↩ Arrival Price", "PNC", "Open", "BPS↩ _Arrival", "Price", "OPP", "Midpoint"
MinFillSize	Minimum number of share per execution for non-displayed liquidity. Rounded down to closest lot size.	Integer	
WoWOrderPct	Max percent of the order on which WoW can work.	Integer	
WoWMode		String	"BLITZ", "DARKSeek", "Seek_↩ Passive", "Seek_Active", "Seek↩ _Aggressive", "Volume_10%", "↩ Volume_20%", "Volume_30%","VWA↩ P_Day", "Patience"

10.2.11.0.0.9 Darkseek (DS)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST

Parameter	Description	Type	Syntax/Values
Relative Limit		Double	Positive and negative values allowed.
Urgency		String	"Passive_Minus", "Passive", "Active", "Active_Plus", "Aggressive"
MinTakeSize	Minimum number of share per execution for displayed liquidity. Rounded down to closest lot size.	Integer	
MinFillSize	Minimum number of share per execution for non-displayed liquidity. Rounded down to closest lot size.	Integer	
IsBuyBack	The algo should engage in SEC Rule 10b-18 restrictions for buy back in US securities.	String	"Yes", "No"
WoWPrice	Would or Work - The price at which the user is willing to complete the order. Used if no WoW reference is specified.	Double	
WowReference	Used with WoW price field. If WoW price is not submitted, the Reference price can be submitted for processing.	String	"Market", "Inside_NBBO_Price", "↩ Arrival Price", "PNC", "Open", "BPS_↩ Arrival", "Price", "OPP", "Midpoint"
WoWOrderPct	Max percent of the order on which WoW can work.	Integer	
WoWMode		String	"BLITZ", "DARKSeek", "Seek_Passive", "Seek_Active", "Seek_Aggressive", "Volume_10%", "Volume_20%", "↩ Volume_30%","VWAP_Day", "Patience"

10.2.11.0.0.10 Post (POST)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
Relative Limit		Double	Positive and negative values allowed.
MaxVolumeRate	Volume limit.	Double	
WoWPrice	Would or Work - The price at which the user is willing to complete the order. Used if no WoW reference is specified.	Double	
WowReference	Used with WoW price field. If WoW price is not submitted, the Reference price can be submitted for processing.	String	"Market", "Inside_NBBO_Price", "↩ Arrival Price", "PNC", "Open", "BPS↩ _Arrival", "Price", "OPP", "Midpoint"
MinTakeSize	Minimum number of share per execution for displayed liquidity. Rounded down to closest lot size.	Integer	
MinFillSize	Minimum number of share per execution for non-displayed liquidity. Rounded down to closest lot size.	Integer	
WoWOrderPct	Max percent of the order on which WoW can work.	Integer	

Parameter	Description	Type	Syntax/Values
WoWMode		String	"BLITZ", "DARKSeek", "Seek_↩ Passive", "Seek_Active", "Seek↩ _Aggressive", "Volume_10%", "↩ Volume_20%", "Volume_30%","VWA↩ P_Day", "Patience"

10.2.11.0.0.11 MultiScale (MS)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
Algo1	The base algo.	String	"Volume_5%", "Volume_10%", "Volume↩ _15%", "Volume_20%", "Volume_25%", "Volume_30%", "DARKSeek", "Seek↩ _Passive", "Seek_Active", "Seek_↩ Aggressive", "BLITZ", "VWAP_Day", "Qty_Scale", "Patience"
MaxQty1	Max quantity for the base algo.	Integer	
Price 2	Trigger price for algo 2 if present.	Double	
Algo2	Underlying algo 2. Must be different than Algo1 or Algo3.	String	"Volume_5%", "Volume_10%", "Volume↩ _15%", "Volume_20%", "Volume_25%", "Volume_30%", "DARKSeek", "Seek↩ _Passive", "Seek_Active", "Seek_↩ Aggressive", "BLITZ", "VWAP_Day", "Qty_Scale", "Patience"
MaxQty2	Max quantity for algo 2	Integer	
Price3	Trigger price for algo 3 if present.	String	
Algo3	Underlying algo 2. Must be different than Algo1 or Algo2.	String	"Volume_5%", "Volume_10%", "Volume↩ _15%", "Volume_20%", "Volume_25%", "Volume_30%", "DARKSeek", "Seek↩ _Passive", "Seek_Active", "Seek_↩ Aggressive", "BLITZ", "VWAP_Day", "Qty_Scale", "Patience"

10.2.11.0.0.12 Opener (OPENER)

Parameter	Description	Type	Syntax/Values
Relative Limit		Double	Positive and negative values allowed.
UnitForOpen	Defines the unit for the open quantity, either shares or percentage.	String	Shares must be in round lots. Percentage must be between 1 and 100.
OpenQty	Determines quantity placed into opening auction. Used unit defined in UnitForOpen.	Double	

Parameter	Description	Type	Syntax/Values
PostOpenStrategy		String	"Volume_5%", "Volume_10%", "↩ Volume_15%", "Volume_20%", "Volume_25%", "Volume_30%", "DARKSeek", "Seek_Passive", "↩ Seek_Active", "Seek_Aggressive", "BLITZ", "VWAP_Day", "Qty_Scale", "Patience"
PostOpenLimit	Absolute limit price for the Post Open strategy.	Double	
PostOpenBenchmark		String	"Inside_NBBO_Price", "Arrival_↩ Price", "PNC", "Open"
BenchmarkOffset	In conjunction with Post Open Benchmark, this sets the relative limit for the strategy.	Integer	Positive or negative value in basis points set as the relative limit from the post open benchmark.

10.2.11.0.0.13 Finale (TOC)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
Urgency		String	"Passive", "Active", "Aggressive"
UnitForClose	The unit to use when defining the close quantity.	String	"Shares", "%_of_Order", "%_of_ADV", "%_of_Expected_Close"
CloseQty	The quantity of the closing auction.	Double	

10.2.11.0.0.14 Portfolio (PORT)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
MaxVolumeRate		Double	
PortfolioId	User defined ID	FixSizeString	
PortfolioLimit	Basis points from arrival	Double	
ExcludeAuctions	Define auction participation	String	"Exclude_Both", "Include_Open", "↩ Include_Close", "Include_Both"
PortfolioUrgency		String	"1", "2", "3", "4", "5"
Style		String	"Cash_Balance", "Beta_Neutral", "IS", "↩ Dark_Only", "Exec_Balance"
CompleteByEOD		String	"Yes", "No"
PriceLimitType		String	"Fixed", "Floating"
Benchmark		String	"Inside_NBBO_Price", "Arrival_Price", "P↩ NC", "Open"
BenchmarkOffset		Integer	Positive or Negative value in basis points.
TrackingIndex		FixSizeString	
MaxOutPerform		Integer	Positive or Negative value in basis points.
MinOutPerform		Integer	Positive or Negative value in basis points.

10.2.11.0.0.15 Patience (PATIENCE)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
MaxVolumeRate	Volume limit.	Double	
Style		Boolean	
WoWPrice	Would or Work - The price at which the user is willing to complete the order. Used if no WoW reference is specified.	Double	
WowReference	Used with WoW price field. If WoW price is not submitted, the Reference price can be submitted for processing.	String	"Market", "Inside_NBBO_Price", "Arrival Price", "PNC", "Open", "BPS_Arrival", "Price", "OPP", "Midpoint"
MinTakeSize	Minimum number of share per execution for displayed liquidity. Rounded down to closest lot size.	Integer	
MinFillSize	Minimum number of share per execution for non-displayed liquidity. Rounded down to closest lot size.	Integer	
WoWOrderPct	Max percent of the order on which WoW can work.	Integer	
WoWMode		String	"BLITZ", "DARKSeek", "Seek_Passive", "Seek_Active", "Seek_Aggressive", "Volume_10%", "Volume_20%", "Volume_30%","VWAP_Day", "Patience"

10.2.11.0.0.16 Pairs - Ratio (RATIO)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
MaxVolumeRate	Volume limit.	Double	
PairID		FixSizeString	
Balance		String	"Share_Balanced", "Cash_Balanced", "Ratio_Balanced"
ExecutionStyle		String	"Active", "TWAP", "Aggressive", "Custom"
LegThreshold		Double	
PairRatio		Double	
PairSpread		Double	
PairSpreadOperator		String	<=, >=

10.2.11.0.0.17 Pairs - Net Return (TR)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
MaxVolumeRate	Volume limit.	Double	
PairID		FixSizeString	
Balance		String	"Share_Balanced", "Cash_Balanced", "Ratio_Balanced"
ExecutionStyle		String	"Active", "TWAP", "Aggressive", "Custom"
LegThreshold		Double	
PairBenchmark		String	"PNC", "Open", "Arrival_Price"
PairSpread		Double	
PairSpreadOperator		String	<=, >=

10.2.11.0.0.18 Pairs - Arb (ARB)

Parameter	Description	Type	Syntax/Values
StartTime	Start time	Time	9:00:00 EST
EndTime	End time	Time	15:00:00 EST
MaxVolumeRate	Volume limit.	Double	
ExecutionStyle		String	"Active", "TWAP", "Aggressive", "Custom"
LegThreshold		Double	
PairID		FixSizeString	
PairObjective		String	"Setup", "Unwind", "Reverse"
PairRatio		Double	
PairCash		Double	
PairSpread		Double	
PairSpreadOperator		String	<=, >=

10.2.11.0.0.19 Trader (TRADER)

Parameter	Description	Type	Syntax/Values
StrategyIntent		String	"Volume_5%", "Volume_10%", "Volume_15%", "Volume_20%", "Volume_25%", "Volume_30%", "DARKSeek", "Seek_Passive", "Seek_Active", "Seek_Aggressive", "BLITZ", "VWAP_Day", "Qty_Scale", "Patience"
ActionType		String	"Halt", "Resume", "Check_Dark", "Take/Hit"
ActionQty		Integer	
ActionPrice		Double	

10.3 Order Management

- **Placing Orders**
- **Modifying Orders**
- **Cancelling Orders**
- **Retrieving currently active orders**
- **Executions and Commissions**
- **Order Limitations**
- **MiFIR Transaction Reporting Fields**

10.3.1 Placing Orders

10.3.1.1 The Next Valid Identifier

Perhaps the most important event received after successfully connecting to the TWS is the **IBApi.EWrapper.nextValidId** (p. **??**), which is also triggered after invoking the **IBApi.EClient.reqIds** method. As its name indicates, the nextValidId event provides the next valid identifier needed to place an order. This identifier is nothing more than the next number in the sequence. This means that if there is a single client application submitting orders to an account, it does not have to obtain a new valid identifier every time it needs to submit a new order. It is enough to increase the last value received from the nextValidId method by one. For example, if the valid identifier for your first API order is 1, the next valid identifier would be 2 and so on.

However if there are multiple client applications connected to one account, it is necessary to use an order ID with new orders which is greater than all previous order IDs returned to the client application in openOrder or orderStatus callbacks. For instance, if the client is set as the Master client, it will automatically receive order status and trade callbacks from orders placed from other clients. In such a case, any orderID used in placeOrder must be greater than the orderIDs returned in these status callbacks. Alternatively if the function reqAllOpenOrders is used by a client, subsequent orders placed by that client must have order IDs greater than the order IDs of all orders returned because of that function call. You can always use the **IBApi.EClient.reqIds** method in the event that your client application loses track of the sequence.

- ```
            //The parameter is always ignored.
            client.reqIds(-1);
  ```
- ```
        //The parameter is always ignored.
        client.reqIds(-1);
  ```
- ```
        'The parameter Is always ignored.
        client.reqIds(-1)
  ```
- ```
    //The parameter is always ignored.
    m_pClient->reqIds(-1);
  ```
- ```
        # The parameter is always ignored.
        self.reqIds(-1)
  ```

The above will result in **IBApi.EWrapper.nextValidId** callback being invoked:

- ```
    public class EWrapperImpl : EWrapper
    {
  ```

...

```
        public virtual void nextValidId(int orderId)
        {
            Console.WriteLine("Next Valid Id: "+orderId);
            NextOrderId = orderId;
        }
```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void nextValidId(int orderId) {
          System.out.println("Next Valid Id: ["+orderId+"]");
          currentOrderId = orderId;
      }
  ```

- ```
      Public Class EWrapperImpl
          Implements EWrapper
  ```

  ...

  ```
          Public Sub nextValidId(orderId As Integer) Implements IBApi.EWrapper.nextValidId
              Console.WriteLine("NextValidId - OrderId [" & orderId & "]")
              nextOrderId = orderId
          End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::nextValidId( OrderId orderId)
  {
      printf("Next Valid Id: %ld\n", orderId);
      m_orderId = orderId;
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def nextValidId(self, orderId: int):
          super().nextValidId(orderId)

          logging.debug("setting nextValidOrderId: %d", orderId)
          self.nextValidOrderId = orderId
  ```

The next valid identifier is persistent between TWS sessions.

If necessary, you can reset the order ID sequence within the API Settings dialogue. Note however that the order sequence Id can only be reset if there are no active API orders.

10.3.1.2 Placing Orders

Orders are submitted via the **IBApi.EClient.placeOrder** method. From the snippet below, note how a vari-able holding the nextValidId is incremented automatically:

- ```
              client.placeOrder(nextOrderId++, ContractSamples.USStock(), OrderSamples.TrailingStopLimit("BUY
    ", 1, 5, 5, 110));
  ```

- ```
        client.placeOrder(nextOrderId++, ContractSamples.USStock(), OrderSamples.LimitOrder("SELL", 1, 50))
    ;
  ```

- ```
        client.placeOrder(increment(nextOrderId), ContractSamples.USStock(),
     OrderSamples.LimitOrder("SELL", 1, 50))
  ```

- ```
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStock(), OrderSamples::LimitOrder("SELL", 1, 50))
    ;
  ```

- ```
        self.simplePlaceOid = self.nextOrderId()
        self.placeOrder(self.simplePlaceOid, ContractSamples.USStock(),
                        OrderSamples.LimitOrder("SELL", 1, 50))
  ```

Immediately after the order was submitted correctly, the TWS will start sending events concerning the order's activity via **IBApi.EWrapper.openOrder** and **IBApi.EWrapper.orderStatus**

- An order can be sent to TWS but not transmitted to the IB server by setting the **IBApi.Order.Transmit** flag in the order class to **False**. Untransmitted orders will only be available within that TWS session (not for other usernames) and will be cleared on restart. Also, they can be cancelled or transmitted from the API but not viewed while they remain in the "untransmitted" state.
  ```

10.3.1.3 The openOrder callback

The **IBApi.EWrapper.openOrder** method delivers an **IBApi.Order** object representing the open order within the system. Additionally, an **IBApi.OrderState** object containing margin and commission information about this particular order.

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
      public virtual void openOrder(int orderId, Contract contract, Order order, OrderState orderState)
      {
          Console.WriteLine("OpenOrder. ID: "+orderId+", "+contract.Symbol+", "+contract.SecType+" @ "+
    contract.Exchange+": "+order.Action+", "+order.OrderType+" "+order.TotalQuantity+", "+orderState.Status);
          if (order.WhatIf)
          {
              Console.WriteLine("What-If. ID: " + orderId +
                  ", InitMarginBefore: " + Util.formatDoubleString(orderState.InitMarginBefore) + ",
     MaintMarginBefore: " + Util.formatDoubleString(orderState.MaintMarginBefore) + " EquityWithLoanBefore: " + Util.
    formatDoubleString(orderState.EquityWithLoanBefore) +
                  ", InitMarginChange: " + Util.formatDoubleString(orderState.InitMarginChange) + ",
     MaintMarginChange: " + Util.formatDoubleString(orderState.MaintMarginChange) + " EquityWithLoanChange: " + Util.
    formatDoubleString(orderState.EquityWithLoanChange) +
                  ", InitMarginAfter: " + Util.formatDoubleString(orderState.InitMarginAfter) + ",
     MaintMarginAfter: " + Util.formatDoubleString(orderState.MaintMarginAfter) + " EquityWithLoanAfter: " + Util.
    formatDoubleString(orderState.EquityWithLoanAfter));
          }
      }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void openOrder(int orderId, Contract contract, Order order,
              OrderState orderState) {
          System.out.println(EWrapperMsgGenerator.openOrder(orderId, contract, order, orderState));
      }
  ```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

  ...

  ```
      Public Sub openOrder(orderId As Integer, contract As IBApi.Contract, order As IBApi.Order,
     orderState As IBApi.OrderState) Implements IBApi.EWrapper.openOrder
          Console.WriteLine("OpenOrder. ID: " & orderId & ", " & contract.Symbol & ", " &
     contract.SecType & " @ " & contract.Exchange &
                            ": " & order.Action & ", " & order.OrderType & " " & order.TotalQuantity & ", " &
     orderState.Status)
          If order.WhatIf = True Then
              Console.WriteLine("What-If. ID: " & orderId &
                  ", InitMarginBefore: " & Util.formatDoubleString(orderState.InitMarginBefore) & ",
     MaintMarginBefore: " & Util.formatDoubleString(orderState.MaintMarginBefore) & " EquityWithLoanBefore: " &
     Util.formatDoubleString(orderState.EquityWithLoanBefore) &
                  ", InitMarginChange: " & Util.formatDoubleString(orderState.InitMarginChange) & ",
     MaintMarginChange: " & Util.formatDoubleString(orderState.MaintMarginChange) & " EquityWithLoanChange: " &
     Util.formatDoubleString(orderState.EquityWithLoanChange) &
                  ", InitMarginAfter: " & Util.formatDoubleString(orderState.InitMarginAfter) & ",
     MaintMarginAfter: " & Util.formatDoubleString(orderState.MaintMarginAfter) & " EquityWithLoanAfter: " &
     Util.formatDoubleString(orderState.EquityWithLoanAfter))
          End If
      End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::openOrder( OrderId orderId, const Contract& contract, const Order& order, const
        OrderState& ostate) {
      printf( "OpenOrder. ID: %ld, %s, %s @ %s: %s, %s, %g, %g, %s, %s\n", orderId, contract.symbol.c_str(),
        contract.secType.c_str(), contract.exchange.c_str(), order.action.c_str(), order.orderType.c_str(), order.
        totalQuantity, order.cashQty == UNSET_DOUBLE ? 0 : order.cashQty, ostate.status.c_str(), order.
        dontUseAutoPriceForHedge ? "true" : "false");
      if (order.whatIf) {
          printf( "What-If. ID: %ld, InitMarginBefore: %s, MaintMarginBefore: %s, EquityWithLoanBefore: %s,
  ```

```
        InitMarginChange: %s, MaintMarginChange: %s, EquityWithLoanChange: %s, InitMarginAfter: %s, MaintMarginAfter:
        %s, EquityWithLoanAfter: %s\n",
              orderId, Utils::formatDoubleString(ostate.initMarginBefore).c_str(), Utils::formatDoubleString(
      ostate.maintMarginBefore).c_str(), Utils::formatDoubleString(ostate.equityWithLoanBefore).c_str(),
              Utils::formatDoubleString(ostate.initMarginChange).c_str(), Utils::formatDoubleString(ostate.
      maintMarginChange).c_str(), Utils::formatDoubleString(ostate.equityWithLoanChange).c_str(),
              Utils::formatDoubleString(ostate.initMarginAfter).c_str(), Utils::formatDoubleString(ostate.
      maintMarginAfter).c_str(), Utils::formatDoubleString(ostate.equityWithLoanAfter).c_str());
      }
  }
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

...

```
    def openOrder(self, orderId: OrderId, contract: Contract, order: Order,
                  orderState: OrderState):
        super().openOrder(orderId, contract, order, orderState)
        print("OpenOrder. ID:", orderId, contract.symbol, contract.secType,
              "@", contract.exchange, ":", order.action, order.orderType,
              order.totalQuantity, orderState.status)
```

10.3.1.4 The orderStatus callback

The **IBApi.EWrapper.orderStatus** method contains all relevant information on the current status of the order execution-wise (i.e. amount filled and pending, filling price, etc.).

- ```
      public class EWrapperImpl : EWrapper
      {
  ```

...

```
       public virtual void orderStatus(int orderId, string status, double filled, double remaining, double
      avgFillPrice, int permId, int parentId, double lastFillPrice, int clientId, string whyHeld, double
      mktCapPrice)
        {
            Console.WriteLine("OrderStatus. Id: "+orderId+", Status: "+status+", Filled"+filled+",
       Remaining: "+remaining
                + ", AvgFillPrice: " + avgFillPrice + ", PermId: " + permId + ", ParentId: " + parentId + "
      , LastFillPrice: " + lastFillPrice + ", ClientId: " + clientId + ", WhyHeld: " + whyHeld + ", MktCapPrice: "
       + mktCapPrice);
        }
```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

...

```
    @Override
    public void orderStatus(int orderId, String status, double filled,
            double remaining, double avgFillPrice, int permId, int parentId,
            double lastFillPrice, int clientId, String whyHeld, double mktCapPrice) {
        System.out.println("OrderStatus. Id: "+orderId+", Status: "+status+", Filled"+filled+", Remaining:
      "+remaining
                +", AvgFillPrice: "+avgFillPrice+", PermId: "+permId+", ParentId: "+parentId+",
       LastFillPrice: "+lastFillPrice+
                ", ClientId: "+clientId+", WhyHeld: "+whyHeld+", MktCapPrice: "+mktCapPrice);
    }
```

- ```
      Public Class EWrapperImpl
          Implements EWrapper
  ```

...

```
       Public Sub orderStatus(orderId As Integer, status As String, filled As Double, remaining As Double,
      avgFillPrice As Double, permId As Integer, parentId As Integer, lastFillPrice As Double, clientId As
      Integer, whyHeld As String, mktCapPrice As Double) Implements IBApi.EWrapper.orderStatus
            Console.WriteLine("OrderStatus. Id: " & orderId & ", Status: " & status & ", Filled" & filled &
      ", Remaining: " & remaining &
                ", AvgFillPrice: " & avgFillPrice & ", PermId: " & permId & ", ParentId: " & parentId & ",
      LastFillPrice: " & lastFillPrice &
                ", ClientId: " & clientId & ", WhyHeld: " & whyHeld & ", mktCapPrice: " & mktCapPrice)
       End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

...
  ```

```
void TestCppClient::orderStatus(OrderId orderId, const std::string& status, double filled,
        double remaining, double avgFillPrice, int permId, int parentId,
        double lastFillPrice, int clientId, const std::string& whyHeld, double mktCapPrice){
    printf("OrderStatus. Id: %ld, Status: %s, Filled: %g, Remaining: %g, AvgFillPrice: %g, PermId: %d,
       LastFillPrice: %g, ClientId: %d, WhyHeld: %s, MktCapPrice: %g\n", orderId, status.c_str(), filled, remaining,
      avgFillPrice, permId, lastFillPrice, clientId, whyHeld.c_str(), mktCapPrice);
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def orderStatus(self, orderId: OrderId, status: str, filled: float,
                      remaining: float, avgFillPrice: float, permId: int,
                      parentId: int, lastFillPrice: float, clientId: int,
                      whyHeld: str, mktCapPrice: float):
          super().orderStatus(orderId, status, filled, remaining,
                              avgFillPrice, permId, parentId, lastFillPrice, clientId, whyHeld, mktCapPrice)
          print("OrderStatus. Id: ", orderId, ", Status: ", status, ", Filled: ", filled,
                ", Remaining: ", remaining, ", AvgFillPrice: ", avgFillPrice,
                ", PermId: ", permId, ", ParentId: ", parentId, ", LastFillPrice: ",
                lastFillPrice, ", ClientId: ", clientId, ", WhyHeld: ",
                whyHeld, ", MktCapPrice: ", mktCapPrice)
  ```

Automatic Order Status Messages (without invoking reqOpenOrders or reqAllOpenOrders)

- Clients with the ID of the client submitting the order will receive order status messages indicating changes in the order status.
- The client with **Master Client ID** (set in TWS/IBG) will receive order status messages for all clients.
- Client ID 0 will receive order status messages for its own (client ID 0) orders and also for orders submitted manually from TWS.

Possible Order States

- **ApiPending** - indicates order has not yet been sent to IB server, for instance if there is a delay in receiving the security definition. Uncommonly received.
- **PendingSubmit** - indicates the order was sent from TWS, but confirmation has not been received that it has been received by the destination. Most commonly because exchange is closed.
- **PendingCancel** - indicates that a request has been sent to cancel an order but confirmation has not been received of its cancellation.
- **PreSubmitted** - indicates that a simulated order type has been accepted by the IB system and that this order has yet to be elected. The order is held in the IB system until the election criteria are met. At that time the order is transmitted to the order destination as specified.
- **Submitted** - indicates that your order has been accepted at the order destination and is working.
- **ApiCancelled** - after an order has been submitted and before it has been acknowledged, an API client client can request its cancellation, producing this state.
- **Cancelled** - indicates that the balance of your order has been confirmed cancelled by the IB system. This could occur unexpectedly when IB or the destination has rejected your order.
- **Filled** - indicates that the order has been completely filled.
- **Inactive** - indicates an order is not working, possible reasons include:
  - it is invalid or triggered an error. A corresponding error code is expected to the error() function.
  - the order is to short shares but the order is being held while shares are being located.
  - an order is placed manually in TWS while the exchange is closed.
  ```

– an order is blocked by TWS due to a precautionary setting and appears there in an untransmitted state

Important notes concerning **IBApi.EWrapper.orderStatus** :

- Typically there are duplicate orderStatus messages with the same information that will be received by a client. This corresponds to messages sent back from TWS, the IB server, or the exchange.
- There are not guaranteed to be orderStatus callbacks for every change in order status. For example with market orders when the order is accepted and executes immediately, there commonly will not be any corresponding orderStatus callbacks. For that reason it is recommended to monitor the **IBApi.EWrapper.execDetails** function in addition to **IBApi.EWrapper.orderStatus** (p. **??**).
- Beginning in API v973.04, a parameter mktCapPrice is included in the orderStatus callback. If an order has been `price-capped`, mktCapPrice will indicate the price at which it has been capped.

10.3.1.5 Attaching Orders

Advanced orders such as **Bracket Orders** or **Hedging** involve attaching child orders to a parent. This can be easily done via the **IBApi.Order.ParentId** attribute by assigning a child order's **IBApi.Order.↩ParentId** to an existing order's **IBApi.Order.OrderId** (p. **??**). When an order is attached to another, the system will keep the child order 'on hold' until its parent fills. Once the parent order is completely filled, its children will automatically become active.

Important: When attaching orders and to prevent accidental executions it is a very good idea to make use of the **IBApi.Order.Transmit** flag as demonstrated in **Bracket Orders**

10.3.2 Modifying Orders

Modification of an API order can be done if the API client is connected to a session of TWS with the same username of TWS and using the same API client ID. The function **IBApi.EClient.placeOrder** can then be called with the same fields as the open order, except for the parameter to modify. This includes the **IBApi.Order.OrderId** (p. **??**), which must match the **IBApi.Order.OrderId** of the **open** order. It is not generally recommended to try to change order fields aside from order price, size, and tif (for DAY -> IOC modifications). To change other parameters, it might be preferable to instead cancel the open order, and create a new one.

- To modify or cancel an individual order placed manually from TWS, it is necessary to connect with client ID 0 and then **bind** the order before attempting to modify it. The process of binding assigns the order an API order ID; prior to binding it will be returned to the API with an **API order ID of 0**. Orders with API order ID 0 **cannot** be modified/cancelled from the API. The function reqOpenOrders binds orders open at that moment which do not already have an API order ID, and the function reqAutoOpenOrders binds future orders automatically. The function reqAllOpenOrders does **not** bind orders.
- To modify API orders when connecting to a different session of TWS (logged in with a different username than used for the original order), it is necessary to first bind the order with client ID 0 in the same manner as manual TWS orders are bound before they can be modified. The binding assignment of API order IDs is independent for each TWS user, so the same order can have different API order IDs for different users. The permID returned in the API Order class which is assigned by TWS can be used to identify an order in an account uniquely.
- Currently (as of TWS version 970) the process of order binding from the API cancels/resubmits an order working on an exchange. This may affect the order's place in the exchange queue. Enhancements are planned to allow for API binding with modification of exchange queue priority.

10.3.3 Cancelling Orders

An order can be cancelled from the API with the functions **IBApi::EClient::cancelOrder** and **IBApi::EClient::reqGlobalCancel** (p. **??**). cancelOrder can only be used to cancel an order that was placed originally by a client with the same client ID (or from TWS for client ID 0). It takes one argument, which is the original order ID.

- `client.cancelOrder(nextOrderId-1);`
- `client.cancelOrder(cancelID);`
- `client.cancelOrder(nextOrderId - 1)`
- `m_pClient->cancelOrder(m_orderId-1);`
- `self.cancelOrder(self.simplePlaceOid)`

IBApi::EClient::reqGlobalCancel will cancel all open orders, regardless of how they were originally placed.

- `client.reqGlobalCancel();`
- `client.reqGlobalCancel();`
- `client.reqGlobalCancel()`
- `m_pClient->reqGlobalCancel();`
- `self.reqGlobalCancel()`

10.3.4 Retrieving currently active orders

As long as an order is active, it is possible to retrieve it using the TWS API. Orders submitted via the TWS API will always be bound to the client application (i.e. client Id) they were submitted from meaning only the submitting client will be able to modify the placed order. Three different methods are provided to allow for maximum flexibility. Active orders will be returned via the **IBApi.EWrapper.openOrder** and **IBApi.EWrapper.orderStatus** methods as already described in **The openOrder callback** and **The orderStatus callback** sections

Note: it is not possible to obtain cancelled or fully filled orders.

10.3.4.1 API client's orders

The **IBApi.EClient.reqOpenOrders** method allows to obtain all active orders submitted by the client ap-plication connected with the exact same client Id with which the order was sent to the TWS. If client 0 invokes reqOpenOrders, it will cause **currently** open orders placed from TWS manually to be 'bound', i.e. assigned an order ID so that they can be modified or cancelled by the API client 0. In the API settings in Global Configuration, is a setting checked by default "Use negative numbers to bind automatic orders" which will specify how manual TWS orders are assigned an API order ID.

- `client.reqOpenOrders();`
- `client.reqOpenOrders();`
- `client.reqOpenOrders()`
- `m_pClient->reqOpenOrders();`
- `self.reqOpenOrders()`

10.3.4.2 All submitted orders

To obtain those orders created via the TWS API regardless of the submitting client application, make use of the **IBApi.EClient.reqAllOpenOrders** function.

- `client.reqAllOpenOrders();`
- `client.reqAllOpenOrders();`
- `client.reqAllOpenOrders()`
- `m_pClient->reqAllOpenOrders();`
- `self.reqAllOpenOrders()`

10.3.4.3 Manually submitted orders

Finally, **IBApi.EClient.reqAutoOpenOrders** can only be invoked by client with ID 0. It will cause future orders placed from TWS to be 'bound', i.e. assigned an order ID such that they can be accessed by the cancel↩Order or placeOrder (for modification) functions by client ID 0.

- `client.reqAutoOpenOrders(true);`
- `client.reqAutoOpenOrders(true);`
- `client.reqAutoOpenOrders(True)`
- `m_pClient->reqAutoOpenOrders(true);`
- `self.reqAutoOpenOrders(True)`

Important: only those applications connecting with client Id 0 will be able to take over manually submitted orders

Through the TWS' API settings it is possible to configure this method's behaviour to some extent. As shown in the image below, manually placed orders can be given a negative order Id which can serve to easily tell manual from API submitted orders. The TWS' tooltip elaborates further:

10.3.4.4 Receiving Order Information

Active orders will be delivered via **The openOrder callback** and **The orderStatus callback** callbacks. When all orders have been sent to the client application you will receive a **IBApi.EWrapper.openOrderEnd** event:

- `public class EWrapperImpl : EWrapper`
  `{`

...

```
        public virtual void openOrder(int orderId, Contract contract, Order order, OrderState orderState)
        {
            Console.WriteLine("OpenOrder. ID: "+orderId+", "+contract.Symbol+", "+contract.SecType+" @ "+
      contract.Exchange+": "+order.Action+", "+order.OrderType+" "+order.TotalQuantity+", "+orderState.Status);
            if (order.WhatIf)
            {
                Console.WriteLine("What-If. ID: " + orderId +
                    ", InitMarginBefore: " + Util.formatDoubleString(orderState.InitMarginBefore) + ",
       MaintMarginBefore: " + Util.formatDoubleString(orderState.MaintMarginBefore) + " EquityWithLoanBefore: " + Util.
      formatDoubleString(orderState.EquityWithLoanBefore) +
                    ", InitMarginChange: " + Util.formatDoubleString(orderState.InitMarginChange) + ",
       MaintMarginChange: " + Util.formatDoubleString(orderState.MaintMarginChange) + " EquityWithLoanChange: " + Util.
      formatDoubleString(orderState.EquityWithLoanChange) +
                    ", InitMarginAfter: " + Util.formatDoubleString(orderState.InitMarginAfter) + ",
       MaintMarginAfter: " + Util.formatDoubleString(orderState.MaintMarginAfter) + " EquityWithLoanAfter: " + Util.
      formatDoubleString(orderState.EquityWithLoanAfter));
            }
        }
```

...

```
        public virtual void orderStatus(int orderId, string status, double filled, double remaining, double
       avgFillPrice, int permId, int parentId, double lastFillPrice, int clientId, string whyHeld, double
      mktCapPrice)
        {
            Console.WriteLine("OrderStatus. Id: "+orderId+", Status: "+status+", Filled"+filled+",
       Remaining: "+remaining
                    + ", AvgFillPrice: " + avgFillPrice + ", PermId: " + permId + ", ParentId: " + parentId + "
      , LastFillPrice: " + lastFillPrice + ", ClientId: " + clientId + ", WhyHeld: " + whyHeld + ", MktCapPrice: "
       + mktCapPrice);
        }
```

...

```
        public virtual void openOrderEnd()
        {
            Console.WriteLine("OpenOrderEnd");
        }
```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

...

```
    @Override
    public void openOrder(int orderId, Contract contract, Order order,
            OrderState orderState) {
        System.out.println(EWrapperMsgGenerator.openOrder(orderId, contract, order, orderState));
    }
```

...

```
    @Override
    public void orderStatus(int orderId, String status, double filled,
            double remaining, double avgFillPrice, int permId, int parentId,
            double lastFillPrice, int clientId, String whyHeld, double mktCapPrice) {
        System.out.println("OrderStatus. Id: "+orderId+", Status: "+status+", Filled"+filled+", Remaining:
      "+remaining
                +", AvgFillPrice: "+avgFillPrice+", PermId: "+permId+", ParentId: "+parentId+",
       LastFillPrice: "+lastFillPrice+
                ", ClientId: "+clientId+", WhyHeld: "+whyHeld+", MktCapPrice: "+mktCapPrice);
    }
```

...

```
    @Override
    public void openOrderEnd() {
        System.out.println("OpenOrderEnd");
    }
```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

...

```
        Public Sub openOrder(orderId As Integer, contract As IBApi.Contract, order As IBApi.Order,
       orderState As IBApi.OrderState) Implements IBApi.EWrapper.openOrder
            Console.WriteLine("OpenOrder. ID: " & orderId & ", " & contract.Symbol & ", " &
       contract.SecType & " @ " & contract.Exchange &
                              ": " & order.Action & ", " & order.OrderType & " " & order.TotalQuantity & ", " &
       orderState.Status)
            If order.WhatIf = True Then
                Console.WriteLine("What-If. ID: " & orderId &
                    ", InitMarginBefore: " & Util.formatDoubleString(orderState.InitMarginBefore) & ",
       MaintMarginBefore: " & Util.formatDoubleString(orderState.MaintMarginBefore) & " EquityWithLoanBefore: " &
       Util.formatDoubleString(orderState.EquityWithLoanBefore) &
                    ", InitMarginChange: " & Util.formatDoubleString(orderState.InitMarginChange) & ",
       MaintMarginChange: " & Util.formatDoubleString(orderState.MaintMarginChange) & " EquityWithLoanChange: " &
       Util.formatDoubleString(orderState.EquityWithLoanChange) &
                    ", InitMarginAfter: " & Util.formatDoubleString(orderState.InitMarginAfter) & ",
       MaintMarginAfter: " & Util.formatDoubleString(orderState.MaintMarginAfter) & " EquityWithLoanAfter: " &
       Util.formatDoubleString(orderState.EquityWithLoanAfter))
            End If
        End Sub
```

...

```
        Public Sub orderStatus(orderId As Integer, status As String, filled As Double, remaining As Double,
       avgFillPrice As Double, permId As Integer, parentId As Integer, lastFillPrice As Double, clientId As
       Integer, whyHeld As String, mktCapPrice As Double) Implements IBApi.EWrapper.orderStatus
            Console.WriteLine("OrderStatus. Id: " & orderId & ", Status: " & status & ", Filled" & filled &
       ", Remaining: " & remaining &
                ", AvgFillPrice: " & avgFillPrice & ", PermId: " & permId & ", ParentId: " & parentId & ",
       LastFillPrice: " & lastFillPrice &
                ", ClientId: " & clientId & ", WhyHeld: " & whyHeld & ", mktCapPrice: " & mktCapPrice)
        End Sub
```

...

```
        Public Sub openOrderEnd() Implements IBApi.EWrapper.openOrderEnd
            Console.WriteLine("OpenOrderEnd")
        End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

...

```
void TestCppClient::openOrder( OrderId orderId, const Contract& contract, const Order& order, const
      OrderState& ostate) {
    printf( "OpenOrder. ID: %ld, %s, %s @ %s: %s, %s, %g, %g, %s, %s\n", orderId, contract.symbol.c_str(),
      contract.secType.c_str(), contract.exchange.c_str(), order.action.c_str(), order.orderType.c_str(), order.
      totalQuantity, order.cashQty == UNSET_DOUBLE ? 0 : order.cashQty, ostate.status.c_str(), order.
      dontUseAutoPriceForHedge ? "true" : "false");
    if (order.whatIf) {
        printf( "What-If. ID: %ld, InitMarginBefore: %s, MaintMarginBefore: %s, EquityWithLoanBefore: %s,
       InitMarginChange: %s, MaintMarginChange: %s, EquityWithLoanChange: %s, InitMarginAfter: %s, MaintMarginAfter:
       %s, EquityWithLoanAfter: %s\n",
            orderId, Utils::formatDoubleString(ostate.initMarginBefore).c_str(), Utils::formatDoubleString(
      ostate.maintMarginBefore).c_str(), Utils::formatDoubleString(ostate.equityWithLoanBefore).c_str(),
            Utils::formatDoubleString(ostate.initMarginChange).c_str(), Utils::formatDoubleString(ostate.
      maintMarginChange).c_str(), Utils::formatDoubleString(ostate.equityWithLoanChange).c_str(),
            Utils::formatDoubleString(ostate.initMarginAfter).c_str(), Utils::formatDoubleString(ostate.
      maintMarginAfter).c_str(), Utils::formatDoubleString(ostate.equityWithLoanAfter).c_str());
    }
}
```

...

```
void TestCppClient::orderStatus(OrderId orderId, const std::string& status, double filled,
       double remaining, double avgFillPrice, int permId, int parentId,
       double lastFillPrice, int clientId, const std::string& whyHeld, double mktCapPrice){
    printf("OrderStatus. Id: %ld, Status: %s, Filled: %g, Remaining: %g, AvgFillPrice: %g, PermId: %d,
      LastFillPrice: %g, ClientId: %d, WhyHeld: %s, MktCapPrice: %g\n", orderId, status.c_str(), filled, remaining,
      avgFillPrice, permId, lastFillPrice, clientId, whyHeld.c_str(), mktCapPrice);
}
```

...

```
void TestCppClient::openOrderEnd() {
    printf( "OpenOrderEnd\n");
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

...

```
    def openOrder(self, orderId: OrderId, contract: Contract, order: Order,
                  orderState: OrderState):
        super().openOrder(orderId, contract, order, orderState)
        print("OpenOrder. ID:", orderId, contract.symbol, contract.secType,
              "@", contract.exchange, ":", order.action, order.orderType,
              order.totalQuantity, orderState.status)
```

...

```
    def orderStatus(self, orderId: OrderId, status: str, filled: float,
                    remaining: float, avgFillPrice: float, permId: int,
                    parentId: int, lastFillPrice: float, clientId: int,
                    whyHeld: str, mktCapPrice: float):
        super().orderStatus(orderId, status, filled, remaining,
                            avgFillPrice, permId, parentId, lastFillPrice, clientId, whyHeld, mktCapPrice)
        print("OrderStatus. Id: ", orderId, ", Status: ", status, ", Filled: ", filled,
              ", Remaining: ", remaining, ", AvgFillPrice: ", avgFillPrice,
              ", PermId: ", permId, ", ParentId: ", parentId, ", LastFillPrice: ",
              lastFillPrice, ", ClientId: ", clientId, ", WhyHeld: ",
              whyHeld, ", MktCapPrice: ", mktCapPrice)
```

...

```
    def openOrderEnd(self):
        super().openOrderEnd()
        print("OpenOrderEnd")
```

10.3.5 Executions and Commissions

When an order is filled either fully or partially, the **IBApi.EWrapper.execDetails** and **IBApi.EWrapper.↩ commissionReport** events will deliver **IBApi.Execution** and **IBApi.CommissionReport** ob-jects. This allows to obtain the full picture of the order's execution and the resulting commissions.

Important: To receive commissions reports for all clients it is necessary to connect as the **Master Client ID** (p. **??**).

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

...

```
        public virtual void execDetails(int reqId, Contract contract, Execution execution)
        {
            Console.WriteLine("ExecDetails. "+reqId+" - "+contract.Symbol+", "+contract.SecType+", "+
      contract.Currency+" - "+execution.ExecId+", "+execution.OrderId+", "+execution.Shares + ", " + execution.
      LastLiquidity);
        }
```

...

```
        public virtual void commissionReport(CommissionReport commissionReport)
        {
            Console.WriteLine("CommissionReport. "+commissionReport.ExecId+" - "+commissionReport.
      Commission+" "+commissionReport.Currency+" RPNL "+commissionReport.RealizedPNL);
        }
```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

...

```
    @Override
    public void execDetails(int reqId, Contract contract, Execution execution) {
        System.out.println("ExecDetails. "+reqId+" - ["+contract.symbol()+"], ["+contract.secType()+"], ["+
      contract.currency()+"], ["+execution.execId()+
                "], ["+execution.orderId()+"], ["+execution.shares()+"]"  + ", [" + execution.lastLiquidity
      () + "]");
    }
```

...

```
    @Override
    public void commissionReport(CommissionReport commissionReport) {
        System.out.println("CommissionReport. ["+commissionReport.m_execId+"] - ["+commissionReport.
      m_commission+"] ["+commissionReport.m_currency+"] RPNL ["+commissionReport.m_realizedPNL+"]");
    }
```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

...

```
        Public Sub execDetails(reqId As Integer, contract As IBApi.Contract, execution As IBApi.Execution)
       Implements IBApi.EWrapper.execDetails
            Console.WriteLine("ExecDetails - ReqId [" & reqId & "] Contract [" & contract.Symbol & ", " &
       contract.SecType &
                              "] Execution [Price: " & execution.Price & ", Exchange: " & execution.Exchange &
       ", Last Liquidity: " & execution.LastLiquidity.ToString() & "]")
        End Sub
```
```

...

```
        Public Sub commissionReport(commissionReport As IBApi.CommissionReport) Implements
       IBApi.EWrapper.commissionReport
            Console.WriteLine("CommissionReport - CommissionReport [" & commissionReport.Commission & " " &
       commissionReport.Currency & "]")
        End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

...

```
void TestCppClient::execDetails( int reqId, const Contract& contract, const Execution& execution) {
    printf( "ExecDetails. ReqId: %d - %s, %s, %s - %s, %ld, %g, %d\n", reqId, contract.symbol.c_str(),
      contract.secType.c_str(), contract.currency.c_str(), execution.execId.c_str(), execution.orderId, execution.
      shares, execution.lastLiquidity);
}
```

...

```
void TestCppClient::commissionReport( const CommissionReport& commissionReport) {
    printf( "CommissionReport. %s - %g %s RPNL %g\n", commissionReport.execId.c_str(), commissionReport.
      commission, commissionReport.currency.c_str(), commissionReport.realizedPNL);
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

...

```
    def execDetails(self, reqId: int, contract: Contract, execution: Execution):
        super().execDetails(reqId, contract, execution)
        print("ExecDetails. ", reqId, contract.symbol, contract.secType, contract.currency,
              execution.execId, execution.orderId, execution.shares, execution.lastLiquidity)
```

...

```
    def commissionReport(self, commissionReport: CommissionReport):
        super().commissionReport(commissionReport)
        print("CommissionReport. ", commissionReport.execId, commissionReport.commission,
              commissionReport.currency, commissionReport.realizedPNL)
```

- Note if a correction to an execution is published it will be received as an additional **IBApi.EWrapper.exec↩ Details** callback with all parameters identical except for the execID in the Execution object. The execID will differ only in the digits after the final period.

10.3.5.1 Requesting Executions

IBApi.Execution and **IBApi.CommissionReport** can be requested on demand via the **IBApi.E↩ Client.reqExecutions** method which receives a **IBApi.ExecutionFilter** object as parameter to obtain only those executions matching the given criteria. An empty **IBApi.ExecutionFilter** object can be passed to obtain all previous executions.

- ```
            client.reqExecutions(10001, new ExecutionFilter());
  ```
- ```
        client.reqExecutions(10001, new ExecutionFilter());
  ```
- ```
        client.reqExecutions(10001, New ExecutionFilter())
  ```
- ```
    m_pClient->reqExecutions(10001, ExecutionFilter());
  ```
- ```
        self.reqExecutions(10001, ExecutionFilter())
  ```

Once all matching executions have been delivered, an **IBApi.EWrapper.execDetailsEnd** event will be triggered.
  ```

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
      public virtual void execDetailsEnd(int reqId)
      {
          Console.WriteLine("ExecDetailsEnd. "+reqId+"\n");
      }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void execDetailsEnd(int reqId) {
          System.out.println("ExecDetailsEnd. "+reqId+"\n");
      }
  ```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

  ...

  ```
      Public Sub execDetailsEnd(reqId As Integer) Implements IBApi.EWrapper.execDetailsEnd
          Console.WriteLine("ExecDetailsEnd - ReqId [" & reqId & "]")
      End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::execDetailsEnd( int reqId) {
      printf( "ExecDetailsEnd. %d\n", reqId);
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def execDetailsEnd(self, reqId: int):
          super().execDetailsEnd(reqId)
          print("ExecDetailsEnd. ", reqId)
  ```

Important: only those executions occurring since the last `IB server restart` for that particular account will be delivered. Older executions will generally not be available via the TWS API.

10.3.6 Order Limitations

Aside from the TWS API's inherent limitation of 50 messages per second implying a maximum of 50 orders per second being sent to the TWS, there are no further API-only limitations. Interactive Brokers however requires its users to monitor their Order Efficiency Ratio (OER) as detailed in the `Considerations for Optimizing Order Efficiency` IBKB article.

Additionally, please note IB allows up to 15 active orders per contract per side per account.
  ```

10.3.7 MiFIR Transaction Reporting Fields

For EEA investment firms required to comply with MiFIR reporting, and who have opted in to Enriched and Delegated Transaction Reporting, we have added four new order attributes to the Order class, and several new presets to TWS and IB Gateway Global Configuration.

New order attributes include:

- **IBApi.Order.Mifid2DecisionMaker** - Used to send "investment decision within the firm" value (if **IBApi.Order.Mifid2DecisionAlgo** is not used).
- **IBApi.Order.Mifid2DecisionAlgo** – Used to send "investment decision within the firm" value (if **IBApi.Order.Mifid2DecisionMaker** is not used).

- **IBApi.Order.Mifid2ExecutionTrader** – Used to send "execution within the firm" value (if **IBApi.Order.Mifid2ExecutionAlgo** is not used).
- **IBApi.Order.Mifid2ExecutionAlgo** - Used to send "execution within the firm" value (if **IBApi.Order.Mifid2ExecutionTrader** is not used).

New TWS and IB Gateway Order Presets can be found in the Orders > MiFIR page of Global Configuration, and include TWS Decision-Maker Defaults, API Decision-Maker Defaults, and Executing Trader/Algo presets.

The following choices are available for the "investment decision within the firm" **IBApi.Order.Mifid2DecisionMaker** and **IBApi.Order.Mifid2DecisionAlgo** attributes:

1. This field does not need to be reported if you are:
   - Using the TWS API to transmit orders, AND
   - The investment decision is always made by the client, AND
   - None of these clients are an EEA investment firm with delegated reporting selected (the "delegated reporting firm").

   You can configure the preset to indicate this via TWS Global Configuration using the Orders > MiFIR page. In this scenario, the orders for the proprietary account will need to be placed via TWS.

2. If you are using the TWS API to transmit orders, and the investment decision is made by a person, or a group of people within a delegated reporting firm, with one person being the primary decision maker:

   - Your TWS API program can, on each order, transmit a decision maker's IB-assigned short code using the field **IBApi.Order.Mifid2DecisionMaker** (p. **??**). You can define persons who can be the decision-makers via IB Account Management. To obtain the short codes that IB assigned to those persons, please contact IB Client Services.

   - If your TWS API program is unable to transmit the above field, and the investment decision is either made by, or approved by, a single person who can be deemed to be the primary investment decision maker, you can pre-configure a default investment decision-maker that will be used for orders where the above fields are not present. You must define the investment decision-maker(s) in IB Account Management, and can then configure the default investment decision-maker in TWS Global Configuration using the Orders > MiFIR page.

3. If you are using the TWS API to transmit orders and the investment decision is made by an algorithm:

   - Your TWS API program can, on each order, transmit a decision maker's IB-assigned short code using the field **IBApi.Order.Mifid2DecisionAlgo** (p. **??**). You can define algorithms that can be the decision-makers via IB Account Management. To obtain the short codes that IB assigned to those persons, please contact IB Client Services.

   - If your TWS API program is unable to transmit the above field, and/or the investment decision is made by a single or primary decision-maker algorithm, you can pre-configure a default investment decision-maker algo that will be used for orders where the above field is not sent. You must define the investment decision-maker(s) in IB Account Management, and can then configure the default investment decision-maker in TWS Global Configuration using the Orders $>$ MiFIR page.

     NOTE: Only ONE investment decision-maker, either a primary person or algorithm, should be provided on an order, or selected as the default.

The following choices are available for "execution within the firm" **IBApi.Order.Mifid2ExecutionTrader** and **IBApi.Order.Mifid2ExecutionAlgo** attributes:

1. No additional information is needed if you are using the TWS API to transmit orders entered in a third-party trading interface, and you are the trader responsible for execution within the firm.
2. If your TWS API program transmits orders to IB automatically without human intervention, please contact **IB Client Services** to register the program or programs with IB as an algo. Only the primary program or algo needs to be registered and identified. You can then configure the default in TWS Global Configuration using the Orders $>$ MiFIR page.
3. Your TWS API program, on each order, can transmit the IB-assigned short code of the algo or person responsible for execution within the firm using the field **IBApi.Order.Mifid2ExecutionAlgo** (for the algorithm) or **IBApi.Order.Mifid2ExecutionTrader** (for the person).

For more information, or to obtain short codes for persons or algos defined in IB Account Management, please contact IB Client Services.

To find out more about the MiFIR transaction reporting obligations, see the `MiFIR Enriched and Delegated Transaction Reporting for EEA Investment Firms` knowledge base article.

10.4 Minimum Price Increment

The minimum increment is the minimum difference between price levels at which a contract can trade. Some trades have constant price increments at all price levels. However some contracts have difference minimum increments on different exchanges on which they trade and/or different minimum increments at different price levels. In the contractDetails class, there is a field 'minTick' which specifies the smallest possible minimum increment encountered on any exchange or price. For complete information about minimum price increment structure, there is the IB Contracts and Securities search site, or the API function reqMarketRule starting in API v973.03 and TWS 966.

The function reqContractDetails when used with a Contract object will return contractDetails object to the contract↩ Details function which has a list of the valid exchanges where the instrument trades. Also within the contractDetails object is a field called marketRuleIDs which has a list of "market rules". A market rule is defined as a rule which defines the minimum price increment given the price. The market rule list returned in contractDetails has a list of market rules in the same order as the list of valid exchanges. In this way, the market rule ID for a contract on a particular exchange can be determined.

- Market rule for forex and forex CFDs indicates default configuration (1/2 and not 1/10 pips). It can be adjusted to 1/10 pips through TWS or IB Gateway Global Configuration.
- Some non-US securities, for instance on the SEHK exchange, have a minimum lot size. This information is not available from the API but can be obtained from the IB Contracts and Securities search page. It will also be indicated in the error message returned from an order which does not conform to the minimum lot size.

With the market rule ID number, the corresponding rule can be found with the API function **IBApi::EClient::req↩ MarketRule** (p. **??**):

```
            client.reqMarketRule(26);
            client.reqMarketRule(240);
```

```
        client.reqMarketRule(26);
        client.reqMarketRule(240);
```

```
        client.reqMarketRule(26)
        client.reqMarketRule(240)
```

```
    m_pClient->reqMarketRule(26);
    m_pClient->reqMarketRule(635);
    m_pClient->reqMarketRule(1388);
```

```
        self.reqMarketRule(26)
        self.reqMarketRule(240)
```

The rule is returned to the function **IBApi::EWrapper::marketRule**

```
        public void marketRule(int marketRuleId, PriceIncrement[] priceIncrements)
        {
            Console.WriteLine("Market Rule Id: " + marketRuleId);
            foreach (var priceIncrement in priceIncrements)
            {
                Console.WriteLine("Low Edge: {0}, Increment: {1}", ((decimal)priceIncrement.LowEdge).
      ToString(), ((decimal)priceIncrement.Increment).ToString());
            }
        }
```

```
    @Override
    public void marketRule(int marketRuleId, PriceIncrement[] priceIncrements) {
        DecimalFormat df = new DecimalFormat("#.#");
        df.setMaximumFractionDigits(340);
        System.out.println("Market Rule Id: " + marketRuleId);
        for (PriceIncrement pi : priceIncrements) {
            System.out.println("Price Increment. Low Edge: " + df.format(pi.lowEdge()) + ", Increment: " +
      df.format(pi.increment()));
        }
    }
```

```
        Public Sub marketRule(marketRuleId As Integer, priceIncrements As PriceIncrement()) Implements
     EWrapper.marketRule
            Console.WriteLine("Market Rule Id:" & marketRuleId)

            For Each priceIncrement In priceIncrements
                Console.WriteLine("LowEdge: " & CDec(priceIncrement.LowEdge) & " Increment: " &
     CDec(priceIncrement.Increment))
            Next
        End Sub
```

```
void TestCppClient::marketRule(int marketRuleId, const std::vector<PriceIncrement> &priceIncrements) {
    printf("Market Rule Id: %d\n", marketRuleId);
    for (unsigned int i = 0; i < priceIncrements.size(); i++) {
        printf("Low Edge: %g, Increment: %g\n", priceIncrements[i].lowEdge, priceIncrements[i].increment);
    }
}
```

- ```
    def marketRule(self, marketRuleId: int, priceIncrements: ListOfPriceIncrements):
        super().marketRule(marketRuleId, priceIncrements)
        print("Market Rule ID: ", marketRuleId)
        for priceIncrement in priceIncrements:
            print("Price Increment. Low Edge: ", priceIncrement.lowEdge,
                  ", Increment: ", priceIncrement.increment)
  ```

- For forex, there is an option in TWS/IB Gateway configuration which allows trading in 1/10 pips instead of 1/5 pips (the default).
- TWS Global Configuration -> Display -> Ticker Row -> Allow Forex trading in 1/10 pips

10.5 Checking Margin Changes

From the API it is possible to check how a specified trade execution is expected to change the account margin requirements for an account in real time. This is done by creating an Order object which has the **IBApi.Order.↩ WhatIf** flag set to true. By default the whatif boolean in Order has a false value, but if set to True in an Order object with is passed to **IBApi.EClient.placeOrder** (p. **??**), instead of sending the order to a destination the IB server it will undergo a credit check for the expected post-trade margin requirement. The estimated post-trade margin requirement is returned to the **IBApi.OrderState** object.

This is equivalent to creating a order ticket in TWS, clicking "Preview", and viewing the information in the "Margin Impact" panel.

See also: `TWS Margin Impact`
  ```

Chapter 10

Streaming Market Data

http://interactivebrokers.github.io/tws-api/market_data.html

It is possible to fetch different kinds market data from the TWS:

- **Top Market Data (Level I)**
- **Market Depth (Level II)**
- **5 Second Real Time Bars**

11.1 Live Market Data

In order to receive real time top-of-book, depth-of-book, or historical market data from the API it is necessary have live market data subscriptions for the requested instruments in TWS. The full list of requirements for real time data:

(1) trading permissions for the specified instruments

(2) a funded account (except with forex and bonds), and

(3) market data subscriptions for the specified username

To subscribe to live market data:

Login to your `Account Management`, navigate to Manage Account -> Trade Configuration -> Market Data and select the relevant packages and/or subscription you wish to subscribe to based on the products you require.

One way to determine which market data subscriptions are required for a given security is to enter the contract into a TWS watchlist and the right-click on the contract to select "Launch Market Data Subscription Manager". This will launch a browser window to the market data subscription page of a subscription covering the indicated instrument.

Alternatively, there is also a "Market Data Assistant" utility for determining market data subscriptions:

Once you have selected the relevant packages, click on the "Continue" button and confirm you have made the right choices in the following screen.

Important: Market Data subscriptions are billable at the full month's rate and will not be pro-rated.

11.2 Sharing Market Data Subscriptions

Market data subscriptions are done at a **TWS user name** level, not per account. This implies that live market data subscriptions need to be purchased per every live TWS user. The only exception to this rule are paper trading users. To share the market data subscriptions simply access your `Account Management` and navigate to Manage Account -> Settings -> Paper Trading where you will be presented to the screen below. It will take up to 24 hours until the market data sharing takes effect.

Important: since your paper trading market data permissions are bound to your live one, you can only obtain live market data on your paper trading user if;

- You have shared the market data subscriptions accordingly as described above.
- You are NOT logged in with your live user name at the same time on a **different** computer.

11.3 Market Data Lines

Whenever a user requests an instrument's real time (top-of-book) market data either from within the TWS or through the TWS API, the user is making use of a market data line. Market data lines therefore represent the **active** market data requests a user has.

Example: To clarify this concept further, let us assume a user has a maxTicker Limit of **ten** market data lines and is already observing the real time data of say **five** stocks within the TWS itself. When the user connects his TWS API client application to the TWS, he then requests the real time market data for another **different five** instruments. At a later point while all 10 requests are still active, the user tries to subscribe to the live real time market data of an eleventh product. Since the user is already making use of ten market data lines (five in the TWS and another five in his client application), the TWS will respond with an error message telling the client application it has reached the maximum number of simultaneous requests. In order to request the market data of the eleventh product, the user will have to cancel at least one of the currently active subscriptions (either within TWS or from his client program.)

By default, every user has a maxTicker Limit of 100 market data lines and as such can obtain the real time market data of up to 100 instruments **simultaneously**. This limit however can be further extended either through the purchase of quote booster packs or by increasing the equity and/or commissions of the user's account. For further details on how to increment the number of market data lines or how is your market data lines' entitlement calculated, please refer to our website's "Market Data Display" section within the `Research, News and Market Data` page.

Note: It is important to understand the concept of market data lines since it has an impact not only on the live real time requests but also for requesting market depth and real time bars.

11.4 Top Market Data (Level I)

Using the TWS API, you can request real time market data for trading and analysis. From the API, market data returned from the function **IBApi.EClient.reqMktData** corresponds to market data displayed in TWS watch-lists. This data is not tick-by-tick but consists of aggregated snapshots taken at intra-second intervals which differ depending on the type of instrument:

- Important to note: Live market data and historical bars are currently not available from the API for the exchanges **OSE** and **SEHKTL**. Only 15 minute delayed streaming data will be available for these exchanges.

Product	Frequency
Stocks, Futures and others	250 ms
US Options	100 ms
FX pairs	5 ms

Requesting Watchlist Data

Receiving Watchlist Data

Cancelling Streaming Data

Component Exchanges

Market Data Types

Beginning in TWS v969 and API v973.04, tick-by-tick data. In TWS, tick-by-tick data is available in the Time & Sales Window. From the API, this corresponds to the function IBApi::EClient::reqTickByTick. Tick-by-tick data can be streamed for up to 5 instruments simultaneously in TWS and the API.

- Real time tick-by-tick data is currently not available for options. Historical tick-by-tick data is available. **Tick-by-Tick Data**

Delayed Streaming Data

11.4.1 Requesting Watchlist Data

Streaming market data values corresponding to data shown in TWS watchlists is available via the **IBApi.EClient.↩ reqMktData** (p. **??**). This data is not tick-by-tick but consists of aggregate snapshots taken several times per second. A set of 'default' tick types are returned by default from a call to **IBApi.EClient.reqMktData** (p. **??**), and additional tick types are available by specifying the corresponding generic tick type in the market data request. Including the generic tick types many, but not all, types of data are available that can be displayed in TWS watchlists by adding additional columns. Each market data request uses a unique identifier (ticker ID) which identifies the returned data:

- ```
  client.reqMktData(1001, ContractSamples.StockComboContract(), string.Empty, false, false, null)
  ;
  ```
- ```
  client.reqMktData(1001, ContractSamples.StockComboContract(), "", false, false, null);
  ```
- ```
  client.reqMktData(1001, ContractSamples.StockComboContract(), String.Empty, False, False, Nothing)
  client.reqMktData(1002, ContractSamples.FuturesOnOptions(), String.Empty, False, False, Nothing)
  ```
- ```
  m_pClient->reqMktData(1001, ContractSamples::StockComboContract(), "", false, false, TagValueListSPtr()
    );
  m_pClient->reqMktData(1002, ContractSamples::OptionWithLocalSymbol(), "", false, false,
    TagValueListSPtr());
  ```
- ```
  self.reqMktData(1000, ContractSamples.USStockAtSmart(), "", False, False, [])
  self.reqMktData(1001, ContractSamples.StockComboContract(), "", True, False, [])
  ```
  ```

11.4.1.1 Generic Tick Types

The most common tick types are delivered automatically after a successful market data request. There are however other tick types available by explicit request: the generic tick types. When invoking **IBApi.EClient.reqMktData** (p. **??**), specific generic ticks can be requested via the **genericTickList** parameter of the function:

- ```
  //Requesting RTVolume (Time & Sales), shortable and Fundamental Ratios generic ticks
  client.reqMktData(1004, ContractSamples.USStock(), "233,236,258", false, false, null);
  ```

- ```
  //Requesting RTVolume (Time & Sales), shortable and Fundamental Ratios generic ticks
  client.reqMktData(1004, ContractSamples.USStock(), "233,236,258", false, false, null);
  ```

- ```
  'Requesting RTVolume (Time & Sales), shortable And Fundamental Ratios generic ticks
  client.reqMktData(1004, ContractSamples.USStock(), "233,236,258", False, False, Nothing)
  ```

- ```
  //Requesting RTVolume (Time & Sales), shortable and Fundamental Ratios generic ticks
  m_pClient->reqMktData(1004, ContractSamples::USStock(), "233,236,258", false, false, TagValueListSPtr()
    );
  ```

- ```
  # Requesting RTVolume (Time & Sales), shortable and Fundamental Ratios generic ticks
  self.reqMktData(1004, ContractSamples.USStock(), "233,236,258", False, False, [])
  ```

For a list of available tick types please refer to **Available Tick Types**

11.4.1.2 Streaming Data Snapshots

With an exchange market data subscription, such as Network A (NYSE), Network B(ARCA), or Network C(NASD↩ AQ) for US stocks, it is possible to request a snapshot of the current state of the market once instead of requesting a stream of updates continuously as market values change. By invoking the **IBApi::EClient::reqMktData** function passing in **true** for the snapshot parameter, the client application will receive the currently available market data once before a **IBApi.EWrapper.tickSnapshotEnd** event is sent 11 seconds later. Snapshot requests can only be made for the default tick types; no generic ticks can be specified. It is important to note that a snapshot request will only return available data over the 11 second span; in many cases values for all default ticks will not be returned. However snapshots are allowed a higher rate limit than the normal 50 messages per second limitation of other EClientSocket functions.

Snapshot request:

- ```
  client.reqMktData(1003, ContractSamples.FutureComboContract(), string.Empty, true, false, null)
  ;
  ```

- ```
  client.reqMktData(1003, ContractSamples.FutureComboContract(), "", true, false, null);
  ```

- ```
  client.reqMktData(1003, ContractSamples.FutureComboContract(), String.Empty, True, False, Nothing)
  ```

- ```
  m_pClient->reqMktData(1003, ContractSamples::FutureComboContract(), "", true, false, TagValueListSPtr()
    );
  ```

- ```
  self.reqMktData(1002, ContractSamples.FutureComboContract(), "", False, False, [])
  ```

End marker reception:

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
  public virtual void tickSnapshotEnd(int tickerId)
  {
      Console.WriteLine("TickSnapshotEnd: "+tickerId);
  }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

```
        @Override
        public void tickSnapshotEnd(int reqId) {
            System.out.println("TickSnapshotEnd: "+reqId);
        }
```

```
    Public Class EWrapperImpl
        Implements EWrapper

...

        Public Sub tickSnapshotEnd(tickerId As Integer) Implements IBApi.EWrapper.tickSnapshotEnd
            Console.WriteLine("TickSnapshotEnd: " & CStr(tickerId))
        End Sub
```

```
class TestCppClient : public EWrapper
{

...

void TestCppClient::tickSnapshotEnd(int reqId) {
    printf( "TickSnapshotEnd: %d\n", reqId);
}
```

```
class TestWrapper(wrapper.EWrapper):

...

    def tickSnapshotEnd(self, reqId: int):
        super().tickSnapshotEnd(reqId)
        print("TickSnapshotEnd:", reqId)
```

11.4.1.3 Regulatory Snapshots

The fifth argument to reqMktData specifies a regulatory snapshot request to US stocks and options. Regulatory snapshots require TWS/IBG **v963** and API **973.02** or higher and specific market data subscriptions.

For stocks, there are individual exchange-specific market data subscriptions necessary to receive streaming quotes. For instance, for NYSE stocks this subscription is known as "Network A", for ARCA/AMEX stocks it is called "↩ Network B" and for NASDAQ stocks it is "Network C". Each subscription is added a la carte and has a separate market data fee.

Alternatively, there is also a "US Securities Snapshot Bundle" subscription which does not provide streaming data but which allows for real time calculated snapshots of US market NBBO prices. By setting the 5th parameter in the function **IBApi::EClient::reqMktData** to **True**, a regulatory snapshot request can be made from the API. The returned value is a calculation of the current market state based on data from all available exchanges.

Important: Each regulatory snapshot made will incur a fee of 0.01 USD to the account. This applies to both live *and* paper accounts.. If the monthly fee for regulatory snapshots reaches the price of a particular 'Network' subscription, the user will automatically be subscribed to that Network subscription for continuous streaming quotes and charged the associated fee for that month. At the end of the month the subscription will be terminated. Each listing exchange will be capped independently and will not be combined across listing exchanges.

```
            // Each regulatory snapshot incurs a 0.01 USD fee
            client.reqMktData(1005, ContractSamples.USStock(), "", false, true, null);
```

```
        // Each regulatory snapshot request incurs a 0.01 USD fee
        client.reqMktData(1014, ContractSamples.USStock(), "", false, true, null);
```

```
        ' Each regulatory snapshot request will incur a fee of 0.01 USD
        ' client.reqMktData(1004, ContractSamples.USStock(), "", False, True, Nothing)
```

```
    // Each regulatory snapshot incurs a fee of 0.01 USD
    m_pClient->reqMktData(1013, ContractSamples::USStock(), "", false, true, TagValueListSPtr());
```

```
        # Each regulatory snapshot request incurs a 0.01 USD fee
        self.reqMktData(1003, ContractSamples.USStock(), "", False, True, [])
```

The following table lists the cost and maximum allocation for regulatory snapshot quotes:

Listed Network Feed	Price per reqSnapshot request	Pro or non-Pro	Max reqSnapshot request
NYSE (Network A/CTA)	0.01 USD	Pro	4500
NYSE (Network A/CTA)	0.01 USD	Non-Pro	150
AMEX (Network B/CTA)	0.01 USD	Pro	2300
AMEX (Network B/CTA)	0.01 USD	Non-Pro	150
NASDAQ (Network C/UTP)	0.01 USD	Pro	2300
NASDAQ (Network C/UTP)	0.01 USD	Non-Pro	150

Requesting regulatory snapshots is subject to pacing limitations:

- If markets are closed, no more than two requests can be made in a 5 second period.
- If markets are open, there can be no more than 5 requests pending for the same contract.

11.4.1.4 Available Tick Types

Note: not all tick types are available for all instruments at all times. If you are not receiving a specific tick type when you think you should see if the tick type in question is available within the TWS itself. Remember the TWS API is only a delivery channel: if the information is not available in the TWS itself first, the TWS will not be able to dispatch it via the API socket.

Tick Name	Tick Id	Description	Delivery Method	Generic tick required
Bid Size	0	Number of contracts or lots offered at the bid price.	**IBApi.EWrapper.tickSize**	-
Bid Price	1	Highest priced bid for the contract.	**IBApi.EWrapper.tickPrice**	-
Ask Price	2	Lowest price offer on the contract.	**IBApi.EWrapper.tickPrice**	-
Ask Size	3	Number of contracts or lots offered at the ask price.	**IBApi.EWrapper.tickSize**	-
Last Price	4	Last price at which the contract traded (does not include some trades in RTVolume).	**IBApi.EWrapper.tickPrice**	-

Tick Name	Tick Id	Description	Delivery Method	Generic tick required
Last Size	5	Number of contracts or lots traded at the last price.	**IBApi.EWrapper.tickSize**	-
High	6	High price for the day.	**IBApi.EWrapper.tickPrice**	-
Low	7	Low price for the day.	**IBApi.EWrapper.tickPrice**	-
Volume	8	Trading volume for the day for the selected contract (US Stocks: multiplier 100).	**IBApi.EWrapper.tickSize**	-
Close Price	9	The last available closing price for the *previous* day. For US Equities, we use corporate action processing to get the closing price, so the close price is adjusted to reflect forward and reverse splits and cash and stock dividends.	**IBApi.EWrapper.tickPrice**	-
Bid Option Computation	10	Computed Greeks and implied volatility based on the underlying stock price and the option bid price. See **Option Greeks**	**IBApi.EWrapper.tickOptionComputation**	-
Ask Option Computation	11	Computed Greeks and implied volatility based on the underlying stock price and the option ask price. See **Option Greeks**	**IBApi.EWrapper.tickOptionComputation**	-
Last Option Computation	12	Computed Greeks and implied volatility based on the underlying stock price and the option last traded price. See **Option Greeks**	**IBApi.EWrapper.tickOptionComputation**	-
Model Option Computation	13	Computed Greeks and implied volatility based on the underlying stock price and the option model price. Correspond to greeks shown in TWS. See **Option Greeks**	**IBApi.EWrapper.tickOptionComputation**	-
Open Tick	14	Today's opening price. The official opening price requires a market data subscription to the native exchange of a contract.	**IBApi.EWrapper.tickPrice**	-

Tick Name	Tick Id	Description	Delivery Method	Generic tick required
Low 13 Weeks	15	Lowest price for the last 13 weeks. For stocks only.	**IBApi.EWrapper.tickPrice**	165
High 13 Weeks	16	Highest price for the last 13 weeks. For stocks only.	**IBApi.EWrapper.tickPrice**	165
Low 26 Weeks	17	Lowest price for the last 26 weeks. For stocks only.	**IBApi.EWrapper.tickPrice**	165
High 26 Weeks	18	Highest price for the last 26 weeks. For stocks only.	**IBApi.EWrapper.tickPrice**	165
Low 52 Weeks	19	Lowest price for the last 52 weeks. For stocks only.	**IBApi.EWrapper.tickPrice**	165
High 52 Weeks	20	Highest price for the last 52 weeks. For stocks only.	**IBApi.EWrapper.tickPrice**	165
Average Volume	21	The average daily trading volume over 90 days. Multiplier of 100. For stocks only.	**IBApi.EWrapper.tickSize**	165
Open Interest	22	(Deprecated, not currently in use) Total number of options that are not closed.	**IBApi.EWrapper.tickSize**	-
Option Historical Volatility	23	The 30-day historical volatility (currently for stocks).	**IBApi.EWrapper.tickGeneric**	104
Option Implied Volatility	24	A prediction of how volatile an underlying will be in the future. The IB 30-day volatility is the at-market volatility estimated for a maturity thirty calendar days forward of the current trading day, and is based on option prices from two consecutive expiration months.	**IBApi.EWrapper.tickGeneric**	106
Option Bid Exchange	25	Not Used.	**IBApi.EWrapper.tickString**	-
Option Ask Exchange	26	Not Used.	**IBApi.EWrapper.tickString**	-
Option Call Open Interest	27	Call option open interest.	**IBApi.EWrapper.tickSize**	101
Option Put Open Interest	28	Put option open interest.	**IBApi.EWrapper.tickSize**	101
Option Call Volume	29	Call option volume for the trading day.	**IBApi.EWrapper.tickSize**	100
Option Put Volume	30	Put option volume for the trading day.	**IBApi.EWrapper.tickSize**	100

Tick Name	Tick Id	Description	Delivery Method	Generic tick required
Index Future Premium	31	The number of points that the index is over the cash index.	**IBApi.EWrapper.tickGeneric**	162
Bid Exchange	32	For stock and options, identifies the exchange(s) posting the bid price. See **Component Exchanges**	**IBApi.EWrapper.tickString**	-
Ask Exchange	33	For stock and options, identifies the exchange(s) posting the ask price. See **Component Exchanges**	**IBApi.EWrapper.tickString**	-
Auction Volume	34	The number of shares that would trade if no new orders were received and the auction were held now.	**IBApi.EWrapper.tickSize**	225
Auction Price	35	The price at which the auction would occur if no new orders were received and the auction were held now- the indicative price for the auction. Typically received after Auction imbalance (tick type 36)	**IBApi.EWrapper.tickPrice**	225
Auction Imbalance	36	The number of unmatched shares for the next auction; returns how many more shares are on one side of the auction than the other. Typically received after Auction Volume (tick type 34)	**IBApi.EWrapper.tickSize**	225
Mark Price	37	The mark price is the current theoretical calculated value of an instrument. Since it is a calculated value, it will typically have many digits of precision.	**IBApi.EWrapper.tickPrice**	221 or 232
Bid EFP Computation	38	Computed EFP bid price	**IBApi.EWrapper.tickEFP**	-
Ask EFP Computation	39	Computed EFP ask price	**IBApi.EWrapper.tickEFP**	-
Last EFP Computation	40	Computed EFP last price	**IBApi.EWrapper.tickEFP**	-
Open EFP Computation	41	Computed EFP open price	**IBApi.EWrapper.tickEFP**	-

Tick Name	Tick Id	Description	Delivery Method	Generic tick required
High EFP Computation	42	Computed high EFP traded price for the day	**IBApi.EWrapper.tickEFP**	-
Low EFP Computation	43	Computed low EFP traded price for the day	**IBApi.EWrapper.tickEFP**	-
Close EFP Computation	44	Computed closing EFP price for previous day	**IBApi.EWrapper.tickEFP**	-
Last Timestamp	45	Time of the last trade (in UNIX time).	**IBApi.EWrapper.tickString**	-
Shortable	46	Describes the level of difficulty with which the contract can be sold short. See **Shortable**	**IBApi.EWrapper.tickGeneric**	236
Fundamental Ratios	47	Provides the available Reuter's Fundamental Ratios. See **Fundamental Ratios** *Note: For some tags the value returned may be -99999.99. This is Reuters' way of indicating data is not available for that tag for the company.* Also see **Fundamental Data** for requesting other fundamental data type using **IBApi.EClient.reqFundamentalData** (p. **??**).	**IBApi.EWrapper.tickString**	258
RT Volume (Time & Sales)	48	Last trade details (Including both "Last" and "Unreportable Last" trades). See **RT Volume**	**IBApi.EWrapper.tickString**	233
Halted	49	Indicates if a contract is halted. See **Halted**	**IBApi.EWrapper.tickGeneric**	-
Bid Yield	50	Implied yield of the bond if it is purchased at the current bid.	**IBApi.EWrapper.tickPrice**	-
Ask Yield	51	Implied yield of the bond if it is purchased at the current ask.	**IBApi.EWrapper.tickPrice**	-
Last Yield	52	Implied yield of the bond if it is purchased at the last price.	**IBApi.EWrapper.tickPrice**	-

Tick Name	Tick Id	Description	Delivery Method	Generic tick required
Custom Option Computation	53	Greek values are based off a user customized price.	**IBApi.EWrapper.tickOptionComputation**	-
Trade Count	54	Trade count for the day.	**IBApi.EWrapper.tickGeneric**	293
Trade Rate	55	Trade count per minute.	**IBApi.EWrapper.tickGeneric**	294
Volume Rate	56	Volume per minute.	**IBApi.EWrapper.tickGeneric**	295
Last RTH Trade	57	Last Regular Trading Hours traded price.	**IBApi.EWrapper.tickPrice**	318
RT Historical Volatility	58	30-day real time historical volatility.	**IBApi.EWrapper.tickGeneric**	411
IB Dividends	59	Contract's dividends. See **IB Dividends**	**IBApi.EWrapper.tickString**	456
Bond Factor Multiplier	60	Not currently implemented.		-
Regulatory Imbalance	61	The imbalance that is used to determine which at-the-open or at-the-close orders can be entered following the publishing of the regulatory imbalance.	**IBApi.EWrapper.tickSize**	-
News	62	Contract's news feed.	**IBApi.EWrapper.tickString**	292
Short-Term Volume 3 Minutes	63	The past three minutes volume. Interpolation may be applied. For stocks only.	**IBApi.EWrapper.tickSize**	595
Short-Term Volume 5 Minutes	64	The past five minutes volume. Interpolation may be applied. For stocks only.	**IBApi.EWrapper.tickSize**	595
Short-Term Volume 10 Minutes	65	The past ten minutes volume. Interpolation may be applied. For stocks only.	**IBApi.EWrapper.tickSize**	595
Delayed Bid	66	Delayed bid price. See **Market Data Types**	**IBApi.EWrapper.tickPrice**	-
Delayed Ask	67	Delayed ask price. See **Market Data Types**	**IBApi.EWrapper.tickPrice**	-
Delayed Last	68	Delayed last traded price. See **Market Data Types**	**IBApi.EWrapper.tickPrice**	-
Delayed Bid Size	69	Delayed bid size. See **Market Data Types**	**IBApi.EWrapper.tickSize**	-

Tick Name	Tick Id	Description	Delivery Method	Generic tick required
Delayed Ask Size	70	Delayed ask size. See **Market Data Types**	**IBApi.EWrapper.tickSize**	-
Delayed Last Size	71	Delayed last size. See **Market Data Types**	**IBApi.EWrapper.tickSize**	-
Delayed High Price	72	Delayed highest price of the day. See **Market Data Types**	**IBApi.EWrapper.tickPrice**	-
Delayed Low Price	73	Delayed lowest price of the day. See **Market Data Types**	**IBApi.EWrapper.tickPrice**	-
Delayed Volume	74	Delayed traded volume of the day. See **Market Data Types**	**IBApi.EWrapper.tickSize**	-
Delayed Close	75	The **prior** day's closing price.	**IBApi.EWrapper.tickPrice**	-
Delayed Open	76	Not typically available	**IBApi.EWrapper.tickPrice**	-
RT Trade Volume	77	Last trade details that excludes "Unreportable Trades". See **RT Trade Volume**	**IBApi.EWrapper.tickString**	375
Creditman mark price	78	Not currently available	**IBApi.EWrapper.tickPrice**	
Creditman slow mark price	79	Slower mark price update used in system calculations	**IBApi.EWrapper.tickPrice**	619
Delayed Bid Option	80	Computed greeks based on delayed bid price. See **Market Data Types** and **Option Greeks**	**IBApi.EWrapper.tickPrice**	
Delayed Ask Option	81	Computed greeks based on delayed ask price. See **Market Data Types** and **Option Greeks**	**IBApi.EWrapper.tickPrice**	
Delayed Last Option	82	Computed greeks based on delayed last price. See **Market Data Types** and **Option Greeks**	**IBApi.EWrapper.tickPrice**	
Delayed Model Option	83	Computed Greeks and model's implied volatility based on delayed stock and option prices.	**IBApi.EWrapper.tickPrice**	

Tick Name	Tick Id	Description	Delivery Method	Generic tick required
Last Exchange	84	Exchange of last traded price	**IBApi.EWrapper.tickString**	
Last Regulatory Time	85	Timestamp (in Unix ms time) of last trade returned with regulatory snapshot	**IBApi.EWrapper.tickString**	
Average Option Volume	87	Average volume of the corresponding option contracts(TWS Build 970+ is required)	**IBApi.EWrapper.tickSize**	105
Delayed Last Timestamp	88	Delayed time of the last trade (in UNIX time) (TWS Build 970+ is required)	**IBApi.EWrapper.tickString**	

11.4.1.4.1 Halted

The **Halted** tick type indicates if a contract has been halted for trading. It can have the following values:

Value	Description
-1	Halted status not available. Usually returned with frozen data.
0	Not halted. This value will **only** be returned if the contract is in a TWS watchlist.
1	General halt. Trading halt is imposed for purely regulatory reasons with/without volatility halt.
2	Volatility halt. Trading halt is imposed by the exchange to protect against extreme volatility.

11.4.1.4.2 Shortable

The shortable tick is an indicative on the amount of shares which can be sold short for the contract:

Range	Description
Value higher than 2.5	There are at least 1000 shares available for short selling.
Value higher than 1.5	This contract will be available for short selling if shares can be located.
1.5 or less	Contract is not available for short selling.

The TWS API cannot send the exact number of shortable shares available. For more detailed information about shortability data (shortable shares, fee rate, rebate rate) available outside of TWS, IB provides an FTP site. For more information, see knowledge base article 2024

11.4.1.4.3 Volume Data

The API reports the current day's volume in several ways. They are summarized as follows:

- Volume tick type 8: The 'native volume'. This includes delayed transactions, busted trades, and combos, but will not update with every tick.
- RTVolume: highest number, includes non-reportable trades such as odd lots, average price and derivative trades.
- RTTradeVolume: only includes 'last' ticks, similar to number also used in charts/historical data.

11.4.1.4.4 RT Volume

The RT Volume tick type corresponds to the TWS' Time & Sales window and contains the last trade's price, size and time along with current day's total traded volume, Volume Weighted Average Price (VWAP) and whether or not the trade was filled by a single market maker.

There is a new setting available starting in TWS v969 which displays tick-by-tick data in the TWS Time & Sales Window. If this setting is checked, it will provide a higher granularity of data than RTVolume.

Example: 701.28;1;1348075471534;67854;701.46918464;true

As volume for US stocks is reported in lots, a volume of 0 reported in RTVolume will typically indicate an odd lot data point (less than 100 shares).

It is important to note that while the TWS Time & Sales Window also has information about trade conditions available with data points, this data is not available through the API. So for instance, the 'unreportable' trade status displayed with points in the Time & Sales Window is not available through the API, and that trade data will appear in the API just as any other data point. As always, an API application needs to exercise caution in responding to single data points.

11.4.1.4.5 RT Trade Volume

The RT Trade Volume is similar to RT Volume, but designed to avoid relaying back "Unreportable Trades" shown in TWS Time&Sales via the API. RT Trade Volume will not contain average price or derivative trades which are included in RTVolume.

11.4.1.4.6 IB Dividends

This tick type provides four different comma-separated elements:

- The sum of dividends for the past 12 months (0.83 in the example below).
- The sum of dividends for the next 12 months (0.92 from the example below).
- The next dividend date (20130219 in the example below).
- The next single dividend amount (0.23 from the example below).

Example: 0.83,0.92,20130219,0.23

To receive dividend information it is sometimes necessary to direct-route rather than smart-route market data requests.

11.4.1.4.7 Delayed Data

Delayed Data support through the API is available with TWS and IBG versions 962 and higher.

To receive delayed data for exchanges without the necessary market data subscriptions for live data, the function call **Market Data Types** is made prior to reqMktData. Quotes for data from 15-20 minutes prior will be streamed back.

Note: API version 9.72 and higher is suggested, but not required, so as to correctly label the delayed tick types (Tick ID 66∼76).

11.4.2 Fundamental Ratios

The Fundamental Ratios tick type is delivered as is from Thomson Reuters. IB does not reformat/re-parse Reuters XML Data, and provides a maximum of 105 tags.

- For older data, it is feasible you may receive less tags (refer to LATESTADATE tag).
- A tag with the value of -99999.99 is an indicator from Reuter's that data is not available for that security.

*Also see **Fundamental Data** for requesting other fundamental data type using **IBApi.EClient.req**↩ **FundamentalData** (*p. **??***).*

See below for all available tag values:

Tag	Description
AATCA	**Total current assets - MRY** This value is the sum of all current assets reported for the most recent fiscal year. ***NOTE:*** *Insurance companies, Banks, and Industrial companies with captive finance subsidiaries do not distinguish between current and long term assets. They report only Total Assets. For these companies this value will be Not Meaningful.*
ACFSHR	**Cash Flow per share - MRY** This is Cash Flow for the most recent fiscal year divided by the Average Shares Outstanding for the same period. Cash Flow is defined as the sum of Income After Taxes minus Preferred Dividends and General Partner Distributions plus Depreciation, Depletion and Amortization.
ADIV5YAVG	**Dividend per share - 5 Yr. Avg.** This is the average of the Annual Dividends Per Share for the last 5 years.
AEBIT	**EBIT - MRY** Earnings before Interest and Tax (EBIT) is computed as Total Revenues for the most recent fiscal year minus Total Operating Expenses plus Operating Interest Expense for the same period. This definition excludes non-operating income and expenses. ***NOTE:*** *This item is only available for Industrial and Utility companies.*
AEBTNORM	**Earnings before taxes Normalized** This is the Income Before Tax number excluding the impact of all unusual/one-time/special charges items for the most recent annual period.
AEPSNORM	**EPS Normalized** This is the Normalized Income Available to Common Stockholders for the most recent annual period divided by the same period's Diluted Weighted Average Shares Outstanding.
AEPSXCLXOR	**EPS excluding extraordinary items - MRY** This is the Adjusted Income Available to Common Stockholders for the most recent fiscal year divided by the most recent fiscal year's Diluted Weighted Average Shares Outstanding.
AFEEPSNTM	**Analyst Forecast of EPS - N12** Analyst Forecast Estimate of EPS for the next twelve months
AFPRD	**Net issuance of debt - MRY** Net issuance of debt for the most recent fiscal year
AFPSS	**Net issuance of stock - MRY** Net issuance of stock for the most recent fiscal year
ALSTD	**Notes payable/short term debt - MRY** Notes payable/short term debt for the most recent fiscal year
ALTCL	**Total current liabilities - MRY** This is the sum of all current and long term liabilities reported for the most recent fiscal year.
ANIACNORM	**Net Income Available to Common, Normalized** This is the annual dollar amount accruing to common shareholders for dividends and retained earnings excluding the impact of all unusual/one-time/special charges items.

Tag	Description
AOTLO	**Cash from operating activities - MRY** Total Cash From Operating Activities represents the sum of Net Income, Depreciation and Amortization, Non-Cash Items, Cash Receipts and Payments, Cash Taxes and Interest Paid, Changes in Working Capital for the most recent fiscal year.
APENORM	**P/E Normalized** This is the Current Price divided by the latest annual Normalized Earnings Per Share value.
APTMGNPCT	**Pretax margin %** This value represents Income Before Taxes for the most recent fiscal year expressed as a percent of Total Revenue for the most recent fiscal year.
AREVPS	**Revenue/share - MRY** This value is the Total Revenue for the most recent fiscal year divided by the Average Diluted Shares Outstanding for the same period.
AROAPCT	**Return on average assets - MRY %** This value is calculated as the Income After Taxes for the most recent fiscal year divided by the Average Total Assets, expressed as a percentage. Average Total Assets is the average of the Total Assets at the beginning and the end of the year.
AROIPCT	**Return on investment - MRY %** This value is the annual Income After Taxes divided by the average Total Long Term Debt, Other Long Term Liabilities, and Shareholders Equity, expressed as a percentage.
ASCEX	**Capital expenditures - MRY** Capital expenditures for the most recent fiscal year
ASFCF	**Financing cash flow items - MRY** Financing cash flow items for the most recent fiscal year
ASICF	**Investing cash flow items - MRY** Investing cash flow items for the most recent fiscal year
ASINN	**Interest exp.(inc.), net-operating - MRY** This is the Total Operating and Non-Operating Interest Expense for the most recent fiscal year. ***NOTE:*** *This item is Not Meaningful for Banks and Insurance companies.*
ASOPI	**Operating income - MRY** This is the amount of profit realized from a business's operations after taking out operating expenses for the most recent fiscal year.
BETA	**Beta** Beta is a measure of a company's common stock price volatility relative to the market. Reuters Beta is the slope of the 60 month regression line of the percentage price change of the stock relative to the percentage price change of the local index. Beta values are not calculated if less than 40 months of pricing is available
CURRENCY	**Currency** The currency in which ratios are denominated.
DIVGRPCT	**Growth rate % - dividend** The Dividend Growth Rate is the compound annual growth rate in dividends per share. DIVGR% is calculated for 3 years whenever 4 years of dividends are available.
EPSCHNGYR	**EPS Change Y/Y %** This value is calculated as the most recent interim period EPS minus the EPS for the same interim period 1 year ago divided by the EPS for the same interim period one year ago, multiplied by 100. ***NOTE:*** *EPS must be positive for both periods. If either EPS value is negative, the result in Not Meaningful (NM).*

Tag	Description
EPSTRENDGR	**EPS growth rate %** This growth rate is the compound annual growth rate of Earnings Per Share Excluding Extraordinary Items and Discontinued Operations over the last 5 years. ***NOTE:*** *If the value for either the most recent year or the oldest year is zero or negative, the growth rate cannot be calculated and a 'NA' (Not Available) code will be used.*
EV_Cur	**Current Enterprise Value** This is the sum of Market Capitalization and Net Debt for the most recent fiscal period. ***NOTE:*** *This item is reported only for Industrial, Utility & Banking companies.*
EV2EBITDA_Cur	**Enterprise Value to EBITDA - TTM** This is the current enterprise value divided by the EBITDA for the most recent trailing twelve months period. ***NOTE:*** *This item is reported only for Industrial and Utility companies.*
Frac52Wk	**52 Week Fraction** This is calculated as (Last - 52 week low) / (52 week high - 52 week low).
LATESTADATE	**Last available date** This value is the date when the ratio data was last updated.
MKTCAP	**Market capitalization** This value is calculated by multiplying the current Price by the current number of Shares Outstanding.
NetDebt_I	**Net Debt, LFI** This value is the sum of all short term debt, and notes payables, Long Term debt, minority interest, and preferred equity minus the total cash and equivalents and short term investments for the most recent interim period. ***NOTE:*** *This item is reported only for Industrial, Utility companies Utility & Banking companies.*
NHIG	**High Price** This price is the highest Price the stock traded at in the last 12 months. This could be an intra-day high.
NLOW	**Low Price** This price is the lowest Price the stock traded at in the last 12 months. This could be an intra-day low.
NPRICE	**Closing Price** This is the close price for the issue from the day it last traded. It is also referred to as the Current Price. Note that some issues may not trade every day, and therefore it is possible for this price to come from a date prior to the last business day.
PEEXCLXOR	**P/E excluding extraordinary items** This ratio is calculated by dividing the current Price by the sum of the Diluted Earnings Per Share from continuing operations BEFORE Extraordinary Items and Accounting Changes over the last four interim periods.
PR13WKPCT	**Price - 13 week price percent change** This is the percentage change in the company's stock price over the last thirteen weeks.
PR1WKPCT	**Price - 1 week price percent change** This is the percentage change in the company's stock price over the last week.
PR2TANBK	**Price/Tang.Book Ratio** This is the Current Price divided by the latest interim period Tangible Book Value Per Share. Tangible Book Value Per Share is defined as Book Value minus Goodwill and Intangible Assets divided by the Shares Outstanding at the end of the interim period.
PR4WKPCT	**Price - 4 week price percent change** This is the percentage change in the company's stock price over the last four weeks.

Tag	Description
PR52WKPCT	**Price - 52 week price percent change** This is the percentage change in the company's stock price over the last fifty two weeks.
PRICE2BK	**Price to Book** This is the Current Price divided by the latest interim period Book Value Per Share.
PRYTDPCTR	**Relative (S&P500) price percent change - YTD** This is the percentage change in price since the close of the last trading day of the previous year as compared to the change in price of the S&P 500 during that same period.
QATCA	**Total current assets - MRQ** This value is the sum of all current assets reported for the most recent interim period. ***NOTE:*** *Insurance companies, Banks, and Industrial companies with captive finance subsidiaries do not distinguish between current and long term assets. They report only Total Assets. For these companies this value will be Not Meaningful.*
QBVPS	**Book Value (Common Equity) per share** This is defined as the Common Shareholder's Equity divided by the Shares Outstanding at the end of the most recent interim period. Book Value is the Total Shareholder's Equity minus Preferred Stock and Redeemable Preferred Stock.
QCASH	**Cash and short term investments - MRQ** This is the Total Cash plus Short Term Investments for the most recent interim period. ***NOTE:*** *This does NOT include cash equivalents that may be reported under long term assets.*
QCSHPS	**Cash per share** This is the Total Cash plus Short Term Investments divided by the Shares Outstanding at the end of the most recent interim period. ***NOTE:*** *This does NOT include cash equivalents that may be reported under long term assets.*
QCURRATIO	**Current ratio** This is the ratio of Total Current Assets for the most recent interim period divided by Total Current Liabilities for the same period. ***NOTE:*** *This item is Not Available (NA) for Banks, Insurance companies and other companies that do not distinguish between current and long term assets and liabilities.*
QEBIT	**EBIT - MRQ** Earnings before Interest and Tax (EBIT) is computed as Total Revenues for the most recent interim period minus Total Operating Expenses plus Operating Interest Expense for the same period. This definition excludes non-operating income and expenses. ***NOTE:*** *This item is only available for Industrial and Utility companies.*
QEBITDA	**EBITD - MRQ** Earnings Before Interest, Taxes, Depreciation and Amortization (EBITDA) is EBIT for the most recent interim period plus the same period's Depreciation and Amortization expenses (from the Statement of Cash Flows). ***NOTE:*** *This item is only available for Industrial and Utility companies.*
QFPRD	**Net issuance of debt - MRQ** Net issuance of debt for the most recent interim period
QFPSS	**Net issuance of stock - MRQ** Net issuance of stock for the most recent interim period
QLSTD	**Notes payable/short term debt - MRQ** Notes payable/short term debt for the most recent interim period
QLTCL	**Total current liabilities - MRQ** This is the sum of all current and long term liabilities reported for the most recent interim period.

Tag	Description
QLTD2EQ	**LT debt/equity** This ratio is the Total Long Term Debt for the most recent interim period divided by Total Shareholder Equity for the same period.
QOTLO	**Cash from operating activities - MRQ** Total Cash From Operating Activities represents the sum of Net Income, Depreciation and Amortization, Non-Cash Items, Cash Receipts and Payments, Cash Taxes and Interest Paid, Changes in Working Capital for the most recent interim period.
QPR2REV	**Price to revenues - MRQ** This is the current Price divided by the Sales Per Share for the most recent interim period. If there is a preliminary earnings announcement for an interim period that has recently ended, the revenue (sales) values from this announcement will be used in calculating the interim Revenues Per Share. ***NOTE:*** *This value has been annualized to make it more readily comparable to the annual and TTM values. Additionally, most Banks and Insurance companies do not report revenues when they announce their preliminary interim financial results in the press. When this happens, the interim value will not be available until the complete interim period filing is released.*
QPRCFPS	**Price to Cash Flow per share - MRQ** This is the current Price divided by Cash Flow Per Share for the most recent interim period. Cash Flow is defined as Income After Taxes minus Preferred Dividends and General Partner Distributions plus Depreciation, Depletion and Amortization.
QQUICKRATI	**Quick ratio** Also known as the Acid Test Ratio, this ratio is defined as Cash plus Short Term Investments plus Accounts Receivable for the most recent interim period divided by the Total Current Liabilities for the same period. ***NOTE:*** *This item is Not Available (NA) for Banks, Insurance companies and other companies that do not distinguish between current and long term assets and liabilities.*
QSCEX	**Capital expenditures - MRQ** Capital expenditures for the most recent interim period
QSFCF	**Financing cash flow items - MRQ** Financing cash flow items for the most recent interim period
QSICF	**Investing cash flow items - MRQ** Investing cash flow items for the most recent interim period
QSINN	**Interest exp.(inc.), net-operating - MRQ** This is the Total Operating and Non-Operating Interest Expense for the most recent interim period. ***NOTE:*** *This item is Not Meaningful for Banks and Insurance companies.*
QSOPI	**Operating income - MRQ** This is the amount of profit realized from a business's operations after taking out operating expenses for the most recent interim period.
QTA	**Total assets - MRQ** This is the sum of all short and long term asset categories reported for the most recent interim period.
QTANBVPS	**Book Value (tangible) per share** This is the interim Tangible Book Value divided by the Shares Outstanding at the end of the most recent interim period. Tangible Book Value is the Book Value minus Goodwill and Intangible Assets for the same period.
QTL	**Total liabilities - MRQ** This is the sum of all current and long term liabilities reported for the most recent interim period.

Tag	Description
QTOTCE	**Total common equity - MRQ** Also referred to as Common Shareholder's Equity, this is the Total Shareholder's Equity as of the most recent interim period Balance Sheet minus Preferred Stock and Redeemable Preferred Stock.
QTOTD2EQ	**Total debt/total equity** This ratio is Total Debt for the most recent interim period divided by Total Shareholder Equity for the same period. ***NOTE:*** *This is Not Meaningful (NM) for banks.*
QTOTLTD	**Total long-term debt - MRQ** This is the sum of all Long Term Debt and Capitalized Lease Obligations for the most recent interim period.
REVCHNGYR	**Revenue Change Y/Y %** This value is calculated as the most recent interim period Sales minus the Sales for the same interim period 1 year ago divided by the Sales for the same interim period one year ago, multiplied by 100.
REVTRENDGR	**Revenue growth rate %** The Five Year Revenue Growth Rate is the annual compounded growth rate of Revenues over the last 5 years.
TTMCFSHR	**Cash Flow per share** This value is the trailing twelve month Cash Flow divided by the trailing twelve month Average Shares Outstanding. Cash Flow is defined as the sum of Income After Taxes minus Preferred Dividends and General Partner Distributions plus Depreciation, Depletion and Amortization.
TTMDIVSHR	**Dividends per share** This is the sum of the Cash Dividends per share paid to common stockholders during the last trailing twelve month period.
TTMEBITD	**EBITD** Earnings Before Interest, Taxes, Depreciation and Amortization (EBITDA) is EBIT for the trailing twelve months plus the same period's Depreciation and Amortization expenses (from the Statement of Cash Flows). ***NOTE:*** *This item is only available for Industrial and Utility companies.*
TTMEBT	**Earnings before taxes** Also known as Pretax Income and Earnings Before Taxes, this is Total Revenue for the most recent TTM period minus Total Expenses plus Non-operating Income (Expenses) for the same period.
TTMEPSCHG	**EPS Change TTM %** This is the percent change in the trailing twelve month EPS as compared to the same trailing twelve month period one year ago. It is calculated as the trailing twelve month EPS minus the trailing twelve month EPS one year ago divided by the trailing twelve month EPS one year ago, multiplied by 100. ***NOTE:*** *If either value has a negative value, the resulting value will be Not Meaningful (NM).*
TTMEPSXCLX	**EPS exc. extr. items** This is the Adjusted Income Available to Common Stockholders for the trailing twelve months divided by the trailing twelve month Diluted Weighted Average Shares Outstanding.
TTMFCF	**Free Cash Flow - TTM** This value is calculated from the Quarterly Statement of Cash Flows. It is calculated as the TTM Cash From Operations minus Capital Expenditures and Dividends Paid for the same period.
TTMFCFSHR	**Free Cash Flow per share - TTM** This is the trailing twelve month Free Cash Flow divided by the trailing twelve month Average Shares Outstanding found on the Income Statement.

Tag	Description
TTMGROSMGN	**Gross Margin %** This value measures the percent of revenue left after paying all direct production expenses. It is calculated as the trailing 12 months Total Revenue minus the trailing 12 months Cost of Goods Sold divided by the trailing 12 months Total Revenue and multiplied by 100. ***NOTE:*** *This item is only available for Industrial and Utility companies.*
TTMINTCOV	**Net Interest coverage - TTM** Also known as Times Interest Earned, this is the ratio of Earnings Before Interest and Taxes for the trailing twelve months divided by the trailing twelve month Interest Expense, Net. ***NOTE:*** *If TTM Total Interest Expense, Net is less or equal 0 (equivalent of Interest Income), then Interest Coverage is Not Meaningful (NM). This item is NM for Banks and Insurance companies.*
TTMINVTURN	**Inventory turnover - TTM** This value measures how quickly the Inventory is sold. It is defined as Cost of Goods Sold for the trailing twelve months divided by Average Inventory. Average Inventory is calculated by adding the Inventory for the 5 most recent quarters and dividing by 5. ***NOTE:*** *This value is Not Meaningful (NM) for Banks and Insurance companies.*
TTMNIAC	**Net Income available to common** This is the trailing twelve month dollar amount accruing to common shareholders for dividends and retained earnings. Income Available to Common Shareholders is calculated as trailing twelve month Income After Taxes plus Minority Interest and Equity in Affiliates plus Preferred Dividends, General Partner Distributions and US GAAP Adjustments. ***NOTE:*** *Any adjustment that is negative (ie. Preferred Stock Dividends) would be subtracted from Income After Taxes.*
TTMNIPEREM	**Net Income/Employee - TTM** This value is the Income After Taxes for the trailing twelve months divided by the number of employees at the end of the last reported fiscal year.
TTMNPMGN	**Net Profit Margin %** Also known as Return on Sales, this value is the Income After Taxes for the trailing twelve months divided by Total Revenue for the same period and is expressed as a percentage. ***NOTE:*** *Most Banks and Finance companies do not report revenues when they announce their preliminary quarterly financial results in the press. When this happens, the trailing twelve month value will not be available (NA).*
TTMOPMGN	**Operating margin %** This value measures the percent of revenues remaining after paying all operating expenses. It is calculated as the trailing 12 months Operating Income divided by the trailing 12 months Total Revenue, multiplied by 100. Operating Income is defined as Total Revenue minus Total Operating Expenses.
TTMPAYRAT	**Payout ratio %** This ratio is the percentage of the Primary/Basic Earnings Per Share Excluding Extraordinary Items paid to common stockholders in the form of cash dividends during the trailing twelve months.
TTMPR2REV	**Price to sales** This is the current Price divided by the Sales Per Share for the trailing twelve months. If there is a preliminary earnings announcement for an interim period that has recently ended, the revenue (sales) values from this announcement will be used in calculating the trailing twelve month revenue per share. ***NOTE:*** *Most Banks and Finance companies do not report revenues when they announce their preliminary interim financial results in the press. When this happens, the trailing twelve month values will not be available (NA) until the complete interim filing is released.*

Tag	Description
TTMPRCFPS	**Price to Cash Flow per share** This is the current Price divided by Cash Flow Per Share for the trailing twelve months. Cash Flow is defined as Income After Taxes minus Preferred Dividends and General Partner Distributions plus Depreciation, Depletion and Amortization.
TTMPRFCFPS	**Price to Free Cash Flow per share - TTM** This is the current Price divided by the trailing twelve month Free Cash Flow Per Share. Free Cash Flow is calculated from the Statement of Cash Flows as Cash From Operations minus Capital Expenditures and Dividends Paid.
TTMPTMGN	**Pretax margin - TTM** This value represents the trailing twelve month Income Before Taxes expressed as a percent of Total Revenue for the same period.
TTMRECTURN	**Receivables turnover - TTM** This is the ratio of Total Revenue for the trailing twelve months divided by Average Accounts Receivables. Average Receivables is calculated by adding the Accounts Receivables for the 5 most recent quarters and dividing by 5. ***NOTE:*** *This item is Not Meaningful (NM) for Banks.*
TTMREV	**Revenue** This is the sum of all revenue (sales) reported for all operating divisions for the most recent TTM period. ***NOTE:*** *Most banks and Insurance companies do not report revenues when they announce their preliminary quarterly financial results in the press. When this happens, the quarterly value will not be available (NA).*
TTMREVCHG	**Revenue Change TTM %** This is the percent change in the trailing twelve month Sales as compared to the same trailing twelve month period one year ago. It is calculated as the trailing twelve month Sales minus the trailing twelve month Sales one year ago divided by the trailing twelve month Sales one year ago, multiplied by 100.
TTMREVPERE	**Revenue/Employee - TTM** This value is the trailing twelve month Total Sales divided by the number of Employees at the end of the last reported fiscal year.
TTMREVPS	**Revenue/share** This value is the trailing twelve month Total Revenue divided by the Average Diluted Shares Outstanding for the trailing twelve months. ***NOTE:*** *Most Banks and Insurance companies do not report revenues when they announce their preliminary quarterly financial results in the press. When this happens, the trailing twelve month values will not be available (NA).*
TTMROAPCT	**Return on average assets %** This value is the Income After Taxes for the trailing twelve months divided by the Average Total Assets, expressed as a percentage. Average Total Assets is calculated by adding the Total Assets for the 5 most recent quarters and dividing by 5.
TTMROEPCT	**Return on average equity %** This value is the Income Available to Common Stockholders for the trailing twelve months divided by the Average Common Equity and is expressed as a percentage. Average Common Equity is calculated by adding the Common Equity for the 5 most recent quarters and dividing by 5.
TTMROIPCT	**Return on investment %** This value is the trailing twelve month Income After Taxes divided by the average Total Long Term Debt, Other Long Term Liabilities and Shareholders Equity, expressed as a percentage.

Tag	Description
YIELD	**Dividend Yield** This value is the current percentage dividend yield based on the present cash dividend rate. It is calculated as the Indicated Annual Dividend divided by the Current Price, multiplied by 100.
YLD5YAVG	**Dividend Yield - 5 Yr. Avg.** This value is the average of the dividend yield over the last 60 months.

11.4.3 Receiving Watchlist Data

11.4.3.1 Receiving Market Data

Real time data is accessed in the API using a subscribe-and-publish model that is also used with other functionality such as retrieving position or account values. After a streaming market data subscription request for a particular contract in IB's database is made, a continuous stream of market data is returned by TWS and separated to different functions in EWrapper depending on the data type (integer vs decimal vs string). Since the subscribe-and-publish model is inherently asynchronous, the returned data is linked to the initial request using a numerical label specified in the request called the request ID (reqID).

After subscribing to the ticker stream, your API client will receive the data supplied by the TWS API server via several methods. In your API client code, you will need to implement these methods to manipulate the data relayed back to the client. Notice how market data callbacks such as **IBApi.EWrapper.tickPrice** and **IBApi.E↩ Wrapper.tickSize** methods contain the request id with which the response can be mapped to its originating request. By default, there are certain 'default tick types' that are always returned. These include fields such as the bid price, ask price, bid size, ask size, etc. There are additional types of data available with the real time data that are requested by specifying certain "generic tick types" in the market data request. For more information, **Generic Tick Types** (p. **??**).

When **IBApi::EWrapper::tickPrice** and **IBApi::EWrapper::tickSize** are reported as -1, this indicates that there is no data currently available. Most commonly this occurs when requesting data from markets that are closed. It can also occur for infrequently trading instruments which do not have open bids or offers at that time of the request. To receive frozen quotes (the last available bid/ask recorded by the system) invoke the function **IBApi.↩ EClient.reqMarketDataType** with argument 2. Alternatively, to receive "delayed frozen" data from tickers without holding market data subscriptions, specify a market data type of 4. Frozen data is exclusive to default tick types- **Generic Tick Types** are not available- and requires market data subscriptions.

For bid, ask, and last data, there will always be a **IBApi::EWrapper::tickSize** callback following each **I↩ BApi::EWrapper::tickPrice** (p. **??**). This is important because with combo contracts, an actively trading contract can have a price of zero. In this case it will have a positive **IBApi::EWrapper::tickSize** value. **IBApi::E↩ Wrapper::tickSize** is also invoked every time there is a change in the size of the last traded price. For that reason, there will be duplicate tickSize events when both the price and size of a trade changes.

-
```
public class EWrapperImpl : EWrapper
{
```

...

```
    public virtual void tickPrice(int tickerId, int field, double price, TickAttrib attribs)
    {
        Console.WriteLine("Tick Price. Ticker Id:"+tickerId+", Field: "+field+", Price: "+price+",
    CanAutoExecute: "+attribs.CanAutoExecute +
            ", PastLimit: " + attribs.PastLimit + ", PreOpen: " + attribs.PreOpen);
    }
```

- The boolean value canAutoExecute in tickPrice is intended to indicate that a quote is available for immediate execution. The boolean value pastLimit indicates whether the bid price is lower than the day's lowest value or the ask price is higher than the highest ask.
- Beginning in API v973.04 the pre-open attribute is available in the API. This attribute indicates that bid/ask quotes released by futures exchanges are in the pre-open period.

```
    public virtual void tickSize(int tickerId, int field, int size)
    {
        Console.WriteLine("Tick Size. Ticker Id:" + tickerId + ", Field: " + field + ", Size: " + size)
;
    }
```

....

```
    public virtual void tickString(int tickerId, int tickType, string value)
    {
        Console.WriteLine("Tick string. Ticker Id:" + tickerId + ", Type: " + tickType + ", Value: " +
value);
    }
```

....

```
    public virtual void tickGeneric(int tickerId, int field, double value)
    {
        Console.WriteLine("Tick Generic. Ticker Id:" + tickerId + ", Field: " + field + ", Value: " +
value);
    }
```

-
```
public class EWrapperImpl implements EWrapper {
```

...

```
@Override
public void tickPrice(int tickerId, int field, double price, TickAttr attribs) {
    System.out.println("Tick Price. Ticker Id:"+tickerId+", Field: "+field+", Price: "+price+",
   CanAutoExecute: "+ attribs.canAutoExecute()
    + ", pastLimit: " + attribs.pastLimit() + ", pre-open: " + attribs.preOpen());
}
```

- The boolean value canAutoExecute in tickPrice is intended to indicate that a quote is available for immediate execution. The boolean value pastLimit indicates whether the bid price is lower than the day's lowest value or the ask price is higher than the highest ask.
- Beginning in API v973.04 the pre-open attribute is available in the API. This attribute indicates that bid/ask quotes released by futures exchanges are in the pre-open period.

```
@Override
public void tickSize(int tickerId, int field, int size) {
    System.out.println("Tick Size. Ticker Id:" + tickerId + ", Field: " + field + ", Size: " + size);
}
```

...

```
@Override
public void tickString(int tickerId, int tickType, String value) {
    System.out.println("Tick string. Ticker Id:" + tickerId + ", Type: " + tickType + ", Value: " +
  value);
}
```

...

```
@Override
public void tickGeneric(int tickerId, int tickType, double value) {
    System.out.println("Tick Generic. Ticker Id:" + tickerId + ", Field: " + TickType.getField(tickType
  ) + ", Value: " + value);
}
```

-
```
Public Class EWrapperImpl
    Implements EWrapper
```

...

```
 Public Sub tickPrice(tickerId As Integer, field As Integer, price As Double, attribs As TickAttrib)
Implements IBApi.EWrapper.tickPrice
     Console.WriteLine("TickPrice - TickerId [" & CStr(tickerId) & "] Field [" &
TickType.getField(field) & "] Price [" & CStr(price) & "] PreOpen [" & attribs.PreOpen & "]")
 End Sub
```

- The boolean value canAutoExecute in tickPrice is intended to indicate that a quote is available for immediate execution. The boolean value pastLimit indicates whether the bid price is lower than the day's lowest value or the ask price is higher than the highest ask.
- Beginning in API v973.04 the pre-open attribute is available in the API. This attribute indicates that bid/ask quotes released by futures exchanges are in the pre-open period.

```
 Public Sub tickSize(tickerId As Integer, field As Integer, size As Integer) Implements
IBApi.EWrapper.tickSize
     Console.WriteLine("Tick Size. Ticker Id:" & CStr(tickerId) & ", Field: " &
TickType.getField(field) & ", Size: " & CStr(size))
 End Sub
```

....

```
 Public Sub tickString(tickerId As Integer, field As Integer, value As String) Implements
IBApi.EWrapper.tickString
     Console.WriteLine("Tick string. Ticker Id:" & CStr(tickerId) & ", Type: " &
TickType.getField(field) & ", Value: " & value)
 End Sub
```

....

```
 Public Sub tickGeneric(tickerId As Integer, field As Integer, value As Double) Implements
IBApi.EWrapper.tickGeneric
     Console.WriteLine("Tick Generic. Ticker Id:" & tickerId & ", Field: " & field & ", Value: " &
value)
 End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::tickPrice( TickerId tickerId, TickType field, double price, const TickAttrib& attribs)
        {
      printf( "Tick Price. Ticker Id: %ld, Field: %d, Price: %g, CanAutoExecute: %d, PastLimit: %d, PreOpen:
         %d\n", tickerId, (int)field, price, attribs.canAutoExecute, attribs.pastLimit, attribs.preOpen);
  }
  ```

  - The boolean value canAutoExecute in tickPrice is intended to indicate that a quote is available for immediate execution. The boolean value pastLimit indicates whether the bid price is lower than the day's lowest value or the ask price is higher than the highest ask.
  - Beginning in API v973.04 the pre-open attribute is available in the API. This attribute indicates that bid/ask quotes released by futures exchanges are in the pre-open period.

  ```
  void TestCppClient::tickSize( TickerId tickerId, TickType field, int size) {
      printf( "Tick Size. Ticker Id: %ld, Field: %d, Size: %d\n", tickerId, (int)field, size);
  }
  ```

  ....

  ```
  void TestCppClient::tickString(TickerId tickerId, TickType tickType, const std::string& value) {
      printf( "Tick String. Ticker Id: %ld, Type: %d, Value: %s\n", tickerId, (int)tickType, value.c_str());
  }
  ```

  ....

  ```
  void TestCppClient::tickGeneric(TickerId tickerId, TickType tickType, double value) {
      printf( "Tick Generic. Ticker Id: %ld, Type: %d, Value: %g\n", tickerId, (int)tickType, value);
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

```
    def tickPrice(self, reqId: TickerId, tickType: TickType, price: float,
                  attrib: TickAttrib):
        super().tickPrice(reqId, tickType, price, attrib)
        print("Tick Price. Ticker Id:", reqId, "tickType:", tickType,
              "Price:", price, "CanAutoExecute:", attrib.canAutoExecute,
              "PastLimit:", attrib.pastLimit, end=' ')
        if tickType == TickTypeEnum.BID or tickType == TickTypeEnum.ASK:
            print("PreOpen:", attrib.preOpen)
        else:
            print()
```

- The boolean value canAutoExecute in tickPrice is intended to indicate that a quote is available for immediate execution. The boolean value pastLimit indicates whether the bid price is lower than the day's lowest value or the ask price is higher than the highest ask.
- Beginning in API v973.04 the pre-open attribute is available in the API. This attribute indicates that bid/ask quotes released by futures exchanges are in the pre-open period.

```
    def tickSize(self, reqId: TickerId, tickType: TickType, size: int):
        super().tickSize(reqId, tickType, size)
        print("Tick Size. Ticker Id:", reqId, "tickType:", tickType, "Size:", size)
```

....

```
    def tickString(self, reqId: TickerId, tickType: TickType, value: str):
        super().tickString(reqId, tickType, value)
        print("Tick string. Ticker Id:", reqId, "Type:", tickType, "Value:", value)
```

....

```
    def tickGeneric(self, reqId: TickerId, tickType: TickType, value: float):
        super().tickGeneric(reqId, tickType, value)
        print("Tick Generic. Ticker Id:", reqId, "tickType:", tickType, "Value:", value)
```

11.4.3.2 Exchange Component Mapping

A market data request is able to return data from multiple exchanges. Beginning in TWS/IBG v**963** and API v**973.02**, after a market data request is made for an instrument covered by market data subscriptions, a message will be sent to function **IBApi::EWrapper::tickReqParams** with information about 'minTick', BBO exchange mapping, and available snapshot permissions.

-
```
        public void tickReqParams(int tickerId, double minTick, string bboExchange, int snapshotPermissions
    )
        {
            Console.WriteLine("id={0} minTick = {1} bboExchange = {2} snapshotPermissions = {3}", tickerId,
    minTick, bboExchange, snapshotPermissions);

            BboExchange = bboExchange;
        }
```

-
```
    @Override
    public void tickReqParams(int tickerId, double minTick, String bboExchange, int snapshotPermissions) {
        System.out.println("Tick req params. Ticker Id:" + tickerId + ", Min tick: " + minTick + ", bbo
    exchange: " + bboExchange + ", Snapshot permissions: " + snapshotPermissions);
    }
```

-
```
        Public Sub tickReqParams(tickerId As Integer, minTick As Double, bboExchange As String,
    snapshotPermissions As Integer) Implements EWrapper.tickReqParams
            Console.WriteLine("id={0} minTick = {1} bboExchange = {2} snapshotPermissions = {3}", tickerId,
    minTick, bboExchange, snapshotPermissions)

            Me.BboExchange = bboExchange
        End Sub
```

-
```
void TestCppClient::tickReqParams(int tickerId, double minTick, const std::string& bboExchange, int
      snapshotPermissions) {
    printf("tickerId: %d, minTick: %g, bboExchange: %s, snapshotPermissions: %u", tickerId, minTick,
      bboExchange.c_str(), snapshotPermissions);

    m_bboExchange = bboExchange;
}
```

- ```
    def tickReqParams(self, tickerId:int, minTick:float,
                      bboExchange:str, snapshotPermissions:int):
        super().tickReqParams(tickerId, minTick, bboExchange, snapshotPermissions)
        print("tickReqParams: ", tickerId, " minTick: ", minTick,
              " bboExchange: ", bboExchange, " snapshotPermissions: ", snapshotPermissions)
  ```

The exchange mapping identifier *bboExchange* will be a symbol such as "a6" which can be used to decode the single letter exchange abbreviations returned to the bidExch, askExch, and lastExch fields by invoking the function **IBApi::EClient::reqSmartComponents** (p. **??**). More information about **Component Exchanges** (p. **??**).

The **minTick** returned to tickReqParams indicates the minimum increment in market data values returned to the API. It can differ from the minTick value in the ContractDetails class. For instance, combos will often have a minimum increment of 0.01 for market data and a minTick of 0.05 for order placement.

11.4.3.3 Re-Routing CFDs

IB does not provide market data for certain types of instruments, such as stock CFDs and forex CFDs. If a stock CFD or forex CFD is entered into a TWS watchlist, TWS will automatically display market data for the underlying ticker and show a 'U' icon next to the instrument name to indicate that the data is for the underlying instrument.

From the API, when level 1 or level 2 market data is requested for a stock CFD or a forex CFD, a callback is made to the functions **IBApi::EWrapper::rerouteMktDataReq** or **IBApi::EWrapper::rerouteMktDepthReq** respectively with details about the underlying instrument in IB's database which does have market data.

- ```
        public void rerouteMktDataReq(int reqId, int conId, string exchange)
        {
            Console.WriteLine("Re-route market data request. Req Id: {0}, ConId: {1}, Exchange: {2}", reqId
      , conId, exchange);
        }
  ```

...

```
        public void rerouteMktDepthReq(int reqId, int conId, string exchange)
        {
            Console.WriteLine("Re-route market depth request. Req Id: {0}, ConId: {1}, Exchange: {2}",
    reqId, conId, exchange);
        }
```

- ```
    @Override
    public void rerouteMktDataReq(int reqId, int conId, String exchange) {
        System.out.println(EWrapperMsgGenerator.rerouteMktDataReq(reqId, conId, exchange));
    }
  ```

...

```
    @Override
    public void rerouteMktDepthReq(int reqId, int conId, String exchange) {
        System.out.println(EWrapperMsgGenerator.rerouteMktDepthReq(reqId, conId, exchange));
    }
```

- ```
        Public Sub rerouteMktDataReq(reqId As Integer, conId As Integer, exchange As String) Implements
      IBApi.EWrapper.rerouteMktDataReq
            Console.WriteLine("Re-route market data request. Req Id: {0}, Con Id: {1}, Exchange: {2}",
      reqId, conId, exchange)
        End Sub
  ```

...

```
        Public Sub rerouteMktDepthReq(reqId As Integer, conId As Integer, exchange As String) Implements
      IBApi.EWrapper.rerouteMktDepthReq
            Console.WriteLine("Re-route market depth request. Req Id: {0}, Con Id: {1}, Exchange: {2}",
      reqId, conId, exchange)
        End Sub
```

- ```
  void TestCppClient::rerouteMktDataReq(int reqId, int conid, const std::string& exchange) {
      printf( "Re-route market data request. ReqId: %d, ConId: %d, Exchange: %s\n", reqId, conid, exchange.
        c_str());
  }
  ```

...
  ```

```
void TestCppClient::rerouteMktDepthReq(int reqId, int conid, const std::string& exchange) {
    printf( "Re-route market depth request. ReqId: %d, ConId: %d, Exchange: %s\n", reqId, conid, exchange.
      c_str());
}
```

-
```
    def rerouteMktDataReq(self, reqId: int, conId: int, exchange: str):
        super().rerouteMktDataReq(reqId, conId, exchange)
        print("Re-route market data request. Req Id: ", reqId,
              ", ConId: ", conId, " Exchange: ", exchange)
```

...

```
    def rerouteMktDepthReq(self, reqId: int, conId: int, exchange: str):
        super().rerouteMktDataReq(reqId, conId, exchange)
        print("Re-route market data request. Req Id: ", reqId,
              ", ConId: ", conId, " Exchange: ", exchange)
```

11.4.4 Cancelling Streaming Data

Once an active market data request is no longer needed, it can be cancelled via the **IBApi::EClient::cancelMkt↩ Data** method. The one argument to cancelMktData is the ticker ID which was specified in original market data request. Cancelling a subscription allows the user to make a subscription to a different contract and remain within the level 1 market data lines allowance.

-
```
            client.cancelMktData(1001);
            client.cancelMktData(1002);
            client.cancelMktData(1003);
            client.cancelMktData(1014);
            client.cancelMktData(1015);
            client.cancelMktData(1016);
```

-
```
        client.cancelMktData(1001);
        client.cancelMktData(1002);
        client.cancelMktData(1003);
        client.cancelMktData(1014);
        client.cancelMktData(1015);
        client.cancelMktData(1016);
```

-
```
        client.cancelMktData(1001)
        client.cancelMktData(1002)
        client.cancelMktData(1003)
        client.cancelMktData(1004)
        client.cancelMktData(1005)
        client.cancelMktData(1014)
        client.cancelMktData(1015)
        client.cancelMktData(1016)
```

-
```
    m_pClient->cancelMktData(1001);
    m_pClient->cancelMktData(1002);
    m_pClient->cancelMktData(1003);
    m_pClient->cancelMktData(1014);
    m_pClient->cancelMktData(1015);
    m_pClient->cancelMktData(1016);
```

-
```
        self.cancelMktData(1000)
        self.cancelMktData(1001)
        self.cancelMktData(1002)
        self.cancelMktData(1003)
```

11.4.5 Component Exchanges

A single data request from the API can receive aggregate quotes from multiple exchanges. With API versions **9.72.18** and TWS **9.62** and higher, the tick types 'bidExch' (tick type 32), 'askExch' (tick type 33), 'lastExch' (tick type 84) are used to identify the source of a quote. To preserve bandwidth, the data returned to these tick types consists of a sequence of capital letters rather than a long list of exchange names for every returned exchange name field. To find the full exchange name corresponding to a single letter code returned in tick types 32, 33, or 84, and API function **IBApi::EClient::reqSmartComponents** is available. **Note:** This function can only be used when the exchange is open.

Different IB contracts have a different exchange map containing the set of exchanges on which they trade. Each exchange map has a different code, such as "a6" or "a9". This exchange mapping code is returned to **IBApi::EWrapper::tickReqParams** immediately after a market data request is made by a user with market data subscriptions. To find a particular map of single letter codes to full exchange names, the function reqSmartComponents is invoked with the exchange mapping code returned to tickReqParams.

- ```
  client.reqSmartComponents(13002, testImpl.BboExchange);
  ```
- ```
  client.reqSmartComponents(1013, "a6");
  ```
- ```
  client.reqSmartComponents(13002, wrapperImpl.BboExchange)
  ```
- ```
  m_pClient->reqSmartComponents(13002, m_bboExchange);
  ```
- ```
  # Requests description of map of single letter exchange codes to full exchange names
  self.reqSmartComponents(1013, "a6")
  ```

For instance, a market data request for the IBKR US contract may return the exchange mapping identifier "a6" to **IBApi.EWrapper.tickReqParams** . Invoking the function **IBApi.EClient.reqSmartComponents** with the symbol "a9" will reveal the list of exchanges offering market data for the IBKR US contract, and their single letter codes. The code for "ARCA" may be "P". In that case if "P" is returned to the exchange tick types, that would indicate the quote was provided by ARCA.

- ```
  public void smartComponents(int reqId, Dictionary<int, KeyValuePair<string, char>> theMap)
  {
      StringBuilder sb = new StringBuilder();

      sb.AppendFormat("==== Smart Components Begin (total={0}) reqId = {1} ====\n", theMap.Count,
  reqId);

      foreach (var item in theMap)
      {
          sb.AppendFormat("bit number: {0}, exchange: {1}, exchange letter: {2}\n", item.Key, item.
  Value.Key, item.Value.Value);
      }

      sb.AppendFormat("==== Smart Components Begin (total={0}) reqId = {1} ====\n", theMap.Count,
  reqId);

      Console.WriteLine(sb);
  }
  ```
- ```
  @Override
  public void smartComponents(int reqId, Map<Integer, Entry<String, Character>> theMap) {
      System.out.println("smart components req id:" + reqId);

      for (Map.Entry<Integer, Entry<String, Character>> item : theMap.entrySet()) {
          System.out.println("bit number: " + item.getKey() +
                  ", exchange: " + item.getValue().getKey() + ", exchange letter: " + item.getValue().
  getValue());
      }
  }
  ```
- ```
  Public Sub smartComponents(reqId As Integer, theMap As Dictionary(Of Integer, KeyValuePair(Of
  String, Char))) Implements EWrapper.smartComponents
      Dim sb As New StringBuilder

      sb.AppendFormat("==== Smart Components Begin (total={0}) reqId = {1} ====={2}", theMap.Count,
  reqId, Environment.NewLine)

      For Each item In theMap
          sb.AppendFormat("bit number: {0}, exchange: {1}, exchange letter: {2}{3}", item.Key,
  ```

```
            item.Value.Key, item.Value.Value, Environment.NewLine)
                Next

                sb.AppendFormat("==== Smart Components Begin (total={0}) reqId = {1} ====={2}", theMap.Count,
            reqId, Environment.NewLine)

                Console.WriteLine(sb)
             End Sub
```

```
• void TestCppClient::smartComponents(int reqId, const SmartComponentsMap& theMap) {
      printf("Smart components: (%lu):\n", theMap.size());

      for (SmartComponentsMap::const_iterator i = theMap.begin(); i != theMap.end(); i++) {
          printf(" bit number: %d exchange: %s exchange letter: %c\n", i->first, std::get<0>(i->second).c_str
        (), std::get<1>(i->second));
      }
  }
```

```
•     def smartComponents(self, reqId:int, map:SmartComponentMap):
          super().smartComponents(reqId, map)
          print("smartComponents: ")
          for exch in map:
              print(exch.bitNumber, ", Exchange Name: ", exch.exchange,
                    ", Letter: ", exch.exchangeLetter)
```

11.4.6 Market Data Types

The API can request Live, Frozen, Delayed and Delayed Frozen market data from Trader Workstation by switching market data type via the **IBApi.EClient.reqMarketDataType** before making a market data request with reqMktData. A successful switch to a different (non-live) market data type for a particular market data request will be indicated by a callback to **IBApi::EWrapper::marketDataType** with the ticker ID of the market data request which is returning a different type of data.

- Beginning in TWS **v970**, a **IBApi.EClient.reqMarketDataType** callback of **1** will occur automatically after invoking reqMktData if the user has live data permissions for the instrument.

Market Data Type	ID	Description
Live	1	Live market data is streaming data relayed back in real time. Market data subscriptions are required to receive live market data.
Frozen	2	Frozen market data is the last data recorded at market close. In TWS, Frozen data is displayed in gray numbers. When you set the market data type to Frozen, you are asking TWS to send the last available quote when there is not one currently available. For instance, if a market is currently closed and real time data is requested, -1 values will commonly be returned for the bid and ask prices to indicate there is no current bid/ask data available. TWS will often show a 'frozen' bid/ask which represents the last value recorded by the system. To receive the last know bid/ask price before the market close, switch to market data type 2 from the API before requesting market data. API frozen data requires TWS/IBG v.962 or higher and the same market data subscriptions necessary for real time streaming data.
Delayed	3	Free, delayed data is 15 - 20 minutes delayed. In TWS, delayed data is displayed in brown background. When you set market data type to delayed, you are telling TWS to automatically switch to delayed market data if the user does not have the necessary real time data subscription. If live data is available a request for delayed data would be ignored by TWS. Delayed market data is returned with delayed **Tick Types** (Tick ID 66∼76). Note: TWS Build 962 or higher is required and API version 9.72.18 or higher is suggested.
Delayed Frozen	4	Requests delayed "frozen" data for a user without market data subscriptions.

- ```
            /*** Switch to live (1) frozen (2) delayed (3) or delayed frozen (4)***/
            client.reqMarketDataType(2);
  ```

- ```
        /*** Switch to live (1) frozen (2) delayed (3) or delayed frozen (4)***/
        client.reqMarketDataType(2);
  ```

- ```
        '*** Switch to live (1) frozen (2) delayed (3) or delayed frozen (4)***/
        client.reqMarketDataType(1)
  ```

- ```
    /*** By default only real-time (1) market data is enabled
         Sending frozen (2) enables frozen market data
         Sending delayed (3) enables delayed market data and disables delayed-frozen market data
         Sending delayed-frozen (4) enables delayed and delayed-frozen market data
         Sending real-time (1) disables frozen, delayed and delayed-frozen market data ***/
    m_pClient->reqMarketDataType(2);
  ```

- ```
        # Switch to live (1) frozen (2) delayed (3) delayed frozen (4).
        self.reqMarketDataType(MarketDataTypeEnum.DELAYED)
  ```

After invoking **IBApi.EClient.reqMarketDataType** to fetch frozen or delayed data, subsequent **IBApi::EClient::reqMktData** requests will trigger **IBApi.EWrapper.marketDataType** as one of the resulting events to indicate the data type (2 or 3) of the returned data.

- ```
    public class EWrapperImpl : EWrapper
    {
  ```

  ...

  ```
        public virtual void marketDataType(int reqId, int marketDataType)
        {
            Console.WriteLine("MarketDataType. "+reqId+", Type: "+marketDataType+"\n");
        }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
    @Override
    public void marketDataType(int reqId, int marketDataType) {
        System.out.println("MarketDataType. ["+reqId+"], Type: ["+marketDataType+"]\n");
    }
  ```

- ```
    Public Class EWrapperImpl
        Implements EWrapper
  ```

  ...

  ```
        Public Sub marketDataType(reqId As Integer, marketDataType As Integer) Implements
       IBApi.EWrapper.marketDataType
            Console.WriteLine("MarketDataType - ReqId [" & reqId & "] MarketDataType [" & marketDataType &
       "]")
        End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::marketDataType(TickerId reqId, int marketDataType) {
      printf( "MarketDataType. ReqId: %ld, Type: %d\n", reqId, marketDataType);
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def marketDataType(self, reqId: TickerId, marketDataType: int):
          super().marketDataType(reqId, marketDataType)
          print("MarketDataType. ", reqId, "Type:", marketDataType)
  ```

11.4.7 Tick-by-Tick Data

Tick-by-tick data corresponding to the data shown in the TWS Time & Sales Window is available starting with TWS v969 and API v973.04. Up to 5 instruments can receive tick-by-tick data at a time.

- The tick type field is case sensitive - it must be BidAsk, Last, AllLast, MidPoint. AllLast has additional trade types such as combos, derivatives, and average price trades which are not included in Last.
- Tick-by-tick data for options is currently only available historically and not in real time.
- Tick-by-tick data is not available for combos.

```
            client.reqTickByTickData(19001, ContractSamples.USStockAtSmart(), "Last", 0, false);
            client.reqTickByTickData(19002, ContractSamples.USStockAtSmart(), "AllLast", 0, false);
            client.reqTickByTickData(19003, ContractSamples.USStockAtSmart(), "BidAsk", 0, true);
            client.reqTickByTickData(19004, ContractSamples.EurGbpFx(), "MidPoint", 0, false);
```

```
        client.reqTickByTickData(19001, ContractSamples.USStockAtSmart(), "Last", 0, false);
        client.reqTickByTickData(19002, ContractSamples.USStockAtSmart(), "AllLast", 0, false);
        client.reqTickByTickData(19003, ContractSamples.USStockAtSmart(), "BidAsk", 0, true);
        client.reqTickByTickData(19004, ContractSamples.EurGbpFx(), "MidPoint", 0, false);
```

```
        client.reqTickByTickData(19001, ContractSamples.EuropeanStock(), "Last", 0, False)
        client.reqTickByTickData(19002, ContractSamples.EuropeanStock(), "AllLast", 0, False)
        client.reqTickByTickData(19003, ContractSamples.EuropeanStock(), "BidAsk", 0, True)
        client.reqTickByTickData(19004, ContractSamples.EurGbpFx(), "MidPoint", 0, False)
```

```
    m_pClient->reqTickByTickData(20005, ContractSamples::EuropeanStock(), "Last", 10, false);
    m_pClient->reqTickByTickData(20006, ContractSamples::EuropeanStock(), "AllLast", 10, false);
    m_pClient->reqTickByTickData(20007, ContractSamples::EuropeanStock(), "BidAsk", 10, false);
    m_pClient->reqTickByTickData(20008, ContractSamples::EurGbpFx(), "MidPoint", 10, true);
```

```
        self.reqTickByTickData(19001, ContractSamples.USStockAtSmart(), "Last", 0, True)
        self.reqTickByTickData(19002, ContractSamples.USStockAtSmart(), "AllLast", 0, False)
        self.reqTickByTickData(19003, ContractSamples.USStockAtSmart(), "BidAsk", 0, True)
        self.reqTickByTickData(19004, ContractSamples.USStockAtSmart(), "MidPoint", 0, False)
```

Depending on the type of data requested, it will be returned to one of the functions **IBApi::EWrapper::historicalTicksLast** (p. **??**), **IBApi::EWrapper::historicalTicksBidAsk** (p. **??**), **IBApi::EWrapper::tickByTick›MidPoint** (p. **??**), or **IBApi::EWrapper::tickByTickAllLast**

```
        public void historicalTicksLast(int reqId, HistoricalTickLast[] ticks, bool done)
        {
            foreach (var tick in ticks)
            {
                Console.WriteLine("Historical Tick Last. Request Id: {0}, Time: {1}, Mask: {2}, Price: {3},
   Size: {4}, Exchange: {5}, Special Conditions: {6}",
                    reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Mask, tick.
  Price, tick.Size, tick.Exchange, tick.SpecialConditions);
            }
        }
```

```
    public void historicalTicksLast(int reqId, List<HistoricalTickLast> ticks, boolean done) {
        for (HistoricalTickLast tick : ticks) {
            System.out.println(EWrapperMsgGenerator.historicalTickLast(reqId, tick.time(), tick.mask(),
  tick.price(), tick.size(), tick.exchange(),
                    tick.specialConditions()));
        }
    }
```

```
        Public Sub historicalTickLast(reqId As Integer, ticks As HistoricalTickLast(), done As Boolean)
    Implements EWrapper.historicalTicksLast
            For Each tick In ticks
                Console.WriteLine("Historical Tick Last. Request Id: {0}, Time: {1}, Mask: {2}, Price: {3},
   Size: {4}, Exchange: {5}, Special Conditions: {6}",
                    reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Mask,
   tick.Price, tick.Size, tick.Exchange, tick.SpecialConditions)
            Next
        End Sub
```

-
```
void TestCppClient::historicalTicksLast(int reqId, const std::vector<HistoricalTickLast>& ticks, bool done)
    {
    for (HistoricalTickLast tick : ticks) {
    std::time_t t = tick.time;
        std::cout << "Historical tick last. ReqId: " << reqId << ", time: " << ctime(&t) << ", mask: " <<
  tick.mask << ", price: "<< tick.price <<
        ", size: " << tick.size << ", exchange: " << tick.exchange << ", special conditions: " << tick.
  specialConditions << std::endl;
    }
}
```

-
```
    def historicalTicksLast(self, reqId: int, ticks: ListOfHistoricalTickLast,
                            done: bool):
        for tick in ticks:
            print("Historical Tick Last. Req Id: ", reqId, ", time: ", tick.time,
                  ", price: ", tick.price, ", size: ", tick.size, ", exchange: ", tick.exchange,
                  ", special conditions:", tick.specialConditions)
```

-
```
        public void historicalTicksBidAsk(int reqId, HistoricalTickBidAsk[] ticks, bool done)
        {
            foreach (var tick in ticks)
            {
                Console.WriteLine("Historical Tick Bid/Ask. Request Id: {0}, Time: {1}, Mask: {2} Price
   Bid: {3}, Price Ask {4}, Size Bid: {5}, Size Ask {6}",
                    reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Mask, tick.
  PriceBid, tick.PriceAsk, tick.SizeBid, tick.SizeAsk);
            }
        }
```

-
```
    @Override
    public void historicalTicksBidAsk(int reqId, List<HistoricalTickBidAsk> ticks, boolean done) {
        for (HistoricalTickBidAsk tick : ticks) {
            System.out.println(EWrapperMsgGenerator.historicalTickBidAsk(reqId, tick.time(), tick.mask(),
  tick.priceBid(), tick.priceAsk(), tick.sizeBid(),
                    tick.sizeAsk()));
        }
    }
```

-
```
        Public Sub historicalTickBidAsk(reqId As Integer, ticks As HistoricalTickBidAsk(), done As Boolean)
    Implements EWrapper.historicalTicksBidAsk
            For Each tick In ticks
                Console.WriteLine("Historical Tick Bid/Ask. Request Id: {0}, Time: {1}, Mask: {2} Price
   Bid: {3}, Price Ask {4}, Size Bid: {5}, Size Ask {6}",
                    reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Mask,
   tick.PriceBid, tick.PriceAsk, tick.SizeBid, tick.SizeAsk)
            Next
        End Sub
```

-
```
void TestCppClient::historicalTicksBidAsk(int reqId, const std::vector<HistoricalTickBidAsk>& ticks, bool
  done) {
    for (HistoricalTickBidAsk tick : ticks) {
    std::time_t t = tick.time;
        std::cout << "Historical tick bid/ask. ReqId: " << reqId << ", time: " << ctime(&t) << ", mask: " <
  < tick.mask << ", price bid: "<< tick.priceBid <<
        ", price ask: "<< tick.priceAsk << ", size bid: " << tick.sizeBid << ", size ask: " << tick.
  sizeAsk << std::endl;
    }
}
```

-
```
    def historicalTicksBidAsk(self, reqId: int, ticks: ListOfHistoricalTickBidAsk,
                              done: bool):
        for tick in ticks:
            print("Historical Tick Bid/Ask. Req Id: ", reqId, ", time: ", tick.time,
                  ", bid price: ", tick.priceBid, ", ask price: ", tick.priceAsk,
                  ", bid size: ", tick.sizeBid, ", ask size: ", tick.sizeAsk)
```

-
```
        public void tickByTickMidPoint(int reqId, long time, double midPoint)
        {
            Console.WriteLine("Tick-By-Tick. Request Id: {0}, TickType: MidPoint, Time: {1}, MidPoint: {2}"
  ,
                reqId, Util.UnixSecondsToString(time, "yyyyMMdd-HH:mm:ss zzz"), midPoint);
        }
```

```java
    @Override
    public void tickByTickMidPoint(int reqId, long time, double midPoint) {
        System.out.println(EWrapperMsgGenerator.tickByTickMidPoint(reqId, time, midPoint));
    }
```

```
        Public Sub tickByTickMidPoint(reqId As Integer, time As Long, midPoint As Double) Implements
       EWrapper.tickByTickMidPoint
            Console.WriteLine("Tick-By-Tick. Request Id: {0}, TickType: MidPoint, Time: {1}, MidPoint:
       {2}",
                reqId, Util.UnixSecondsToString(time, "yyyyMMdd-HH:mm:ss zzz"), midPoint)
        End Sub
```

```
void TestCppClient::tickByTickMidPoint(int reqId, time_t time, double midPoint) {
    printf("Tick-By-Tick. ReqId: %d, TickType: MidPoint, Time: %s, MidPoint: %g\n", reqId, ctime(&time),
      midPoint);
}
```

```
    @iswrapper
    def tickByTickMidPoint(self, reqId: int, time: int, midPoint: float):
        super().tickByTickMidPoint(reqId, time, midPoint)
        print("Midpoint. Req Id: ", reqId,
              " Time: ", datetime.datetime.fromtimestamp(time).strftime("%Y%m%d %H:%M:%S"),
              " MidPoint: ", midPoint)
```

```
        public void tickByTickAllLast(int reqId, int tickType, long time, double price, int size,
      TickAttrib attribs, string exchange, string specialConditions)
        {
            Console.WriteLine("Tick-By-Tick. Request Id: {0}, TickType: {1}, Time: {2}, Price: {3}, Size:
       {4}, Exchange: {5}, Special Conditions: {6}, PastLimit: {7}, Unreported: {8}",
                reqId, tickType == 1 ? "Last" : "AllLast", Util.UnixSecondsToString(time, "
      yyyyMMdd-HH:mm:ss zzz"), price, size, exchange, specialConditions, attribs.PastLimit, attribs.Unreported);
        }
```

```
    @Override
    public void tickByTickAllLast(int reqId, int tickType, long time, double price, int size, TickAttr
      attribs,
            String exchange, String specialConditions) {
        System.out.println(EWrapperMsgGenerator.tickByTickAllLast(reqId, tickType, time, price, size,
      attribs, exchange, specialConditions));
    }
```

```
        Public Sub tickByTickAllLast(reqId As Integer, tickType As Integer, time As Long, price As Double,
       size As Integer, attribs As TickAttrib, exchange As String, specialConditions As String) Implements
       EWrapper.tickByTickAllLast
            Dim tickTypeStr As String
            If tickType = 1 Then
                tickTypeStr = "Last"
            Else
                tickTypeStr = "AllLast"
            End If
            Console.WriteLine("Tick-By-Tick. Request Id: {0}, TickType: {1}, Time: {2}, Price: {3}, Size:
       {4}, Exchange: {5}, Special Conditions: {6}, PastLimit: {7}, Unreported: {8}",
                reqId, tickTypeStr, Util.UnixSecondsToString(time, "yyyyMMdd-HH:mm:ss zzz"), price, size,
       exchange, specialConditions, attribs.PastLimit, attribs.Unreported)
        End Sub
```

```
void TestCppClient::tickByTickAllLast(int reqId, int tickType, time_t time, double price, int size, const
       TickAttrib& attribs, const std::string& exchange, const std::string& specialConditions) {
    printf("Tick-By-Tick. ReqId: %d, TickType: %s, Time: %s, Price: %g, Size: %d, PastLimit: %d,
       Unreported: %d, Exchange: %s, SpecialConditions:%s\n",
        reqId, (tickType == 1 ? "Last" : "AllLast"), ctime(&time), price, size, attribs.pastLimit, attribs.
      unreported, exchange.c_str(), specialConditions.c_str());
}
```

```
    def tickByTickAllLast(self, reqId: int, tickType: int, time: int, price: float,
                          size: int, attribs: TickAttrib, exchange: str,
                          specialConditions: str):
        super().tickByTickAllLast(reqId, tickType, time, price, size, attribs,
                                  exchange, specialConditions)
        if tickType == 1:
            print("Last.", end='')
        else:
            print("AllLast.", end='')
        print(" ReqId: ", reqId,
              " Time: ", datetime.datetime.fromtimestamp(time).strftime("%Y%m%d %H:%M:%S"),
              " Price: ", price, " Size: ", size, " Exch: " , exchange,
              "Spec Cond: ", specialConditions, end='')
        if attribs.pastLimit:
            print(" pastLimit ", end='')
        if attribs.unreported:
            print(" unreported", end='')
        print()
```

Tick-by-tick subscriptions can be cancelled using the function **IBApi::EClient::cancelTickByTickData**

```
            client.cancelTickByTickData(19001);
            client.cancelTickByTickData(19002);
            client.cancelTickByTickData(19003);
            client.cancelTickByTickData(19004);
```

```
        client.cancelTickByTickData(19001);
        client.cancelTickByTickData(19002);
        client.cancelTickByTickData(19003);
        client.cancelTickByTickData(19004);
```

```
        client.cancelTickByTickData(19001)
        client.cancelTickByTickData(19002)
        client.cancelTickByTickData(19003)
        client.cancelTickByTickData(19004)
```

```
    m_pClient->cancelTickByTickData(20001);
    m_pClient->cancelTickByTickData(20002);
    m_pClient->cancelTickByTickData(20003);
    m_pClient->cancelTickByTickData(20004);
```

```
        self.cancelTickByTickData(19001)
        self.cancelTickByTickData(19002)
        self.cancelTickByTickData(19003)
        self.cancelTickByTickData(19004)
```

11.4.8 Delayed Streaming Data

10-15 minute delayed streaming data is available for many types of instruments without market data subscriptions. By default, the API is in the real time market data mode, so the function **IBApi::EClient::reqMktData** will request real time data. To switch to delayed streaming data, the function **IBApi::EClient::reqMarketDataType** must be invoked with a parameter of 3 (for delayed) or 4 (for delayed-frozen) quotes, see also **Market Data Types** (p. **??**). Subsequent calls to **IBApi::EClient::reqMktData** will then result in requests for delayed data which will be returned to the delayed tick types **Available Tick Types** (p. **??**). If live data is available, it will always be returned instead of delayed data.

- Delayed data has a limited number of generic tick types that can be specified: 101, 106, 165, 221, 232, 236, 258. See also **Available Tick Types**
- It is important to note that historical data still requires market data subscriptions **Historical Market Data**

11.5 Market Depth (Level II)

Market depth data, also known as level II, represent's an instrument's order book. Via the TWS API it is possible to obtain this information thanks to the **IBApi.EClient.reqMarketDepth** function. Unlike **Top Market Data (Level I)** (p. **??**), market depth data is sent without sampling nor filtering, however we cannot guarantee that every price quoted for a particular security will be displayed when you invoke **IBApi.EClient.reqMarketDepth** (p. **??**). Market depth requests must be routed to a particular exchange - it is **not** possible to request a smart-routed aggre-gate quote as with level 1 data.

11.5.1 Requesting

The **IBApi.EClient.reqMarketDepth** method receives a request identifier (or ticker Id) with which to identify the incoming data, the **IBApi.Contract** for which we want to pull this information and the number of rows or depth level that is required. In case the market depth is smaller than the requested number of rows, the TWS will simply return the available entries.

- `client.reqMarketDepth(2001, ContractSamples.EurGbpFx(), 5, null);`
- `client.reqMktDepth(2001, ContractSamples.EurGbpFx(), 5, null);`
- `client.reqMarketDepth(2001, ContractSamples.EurGbpFx(), 5, Nothing)`
- `m_pClient->reqMktDepth(2001, ContractSamples::EurGbpFx(), 5, TagValueListSPtr());`
- ```
  self.reqMktDepth(2101, ContractSamples.USStock(), 5, [])
  self.reqMktDepth(2001, ContractSamples.EurGbpFx(), 5, [])
  ```

11.5.2 Exchanges supplying Market Makers

Market depth will be returned via the **IBApi.EWrapper.updateMktDepth** and **IBApi.EWrapper.update↩ MktDepthL2** callback. The two functions differentiate in that, for exchanges have additional Market Makers than itself, market depth data will be relayed back through **IBApi.EWrapper.updateMktDepthL2** (p. **??**), otherwise **IBApi.EWrapper.updateMktDepth** (p. **??**). For example, ARCA only has ARCA itself as a Market Maker. Therefore when requesting market depth data from ARCA, the data will be relayed back via **IBApi.EWrapper.updateMkt↩ Depth** (p. **??**). On the other hand, exchange like ISLAND (ECN for NASDAQ) has additional Market Makers, so the data will be relayed back via **IBApi.EWrapper.updateMktDepthL2** (p. **??**), where the Market Maker information is directly available as a parameter of the call back function.

To check which market depth subscriptions provide market maker information, the function **IBApi::EClient::req↩ MktDepthExchanges** can be invoked. It will return a list of exchanges from where market depth is available if the user has the appropriate market data subscription.

- `client.reqMktDepthExchanges();`
- `client.reqMktDepthExchanges();`
- `client.reqMktDepthExchanges()`
- `m_pClient->reqMktDepthExchanges();`
- `self.reqMktDepthExchanges()`

API 'Exchange' fields for which a market depth request would return market maker information and result in a callback to **IBApi::EWrapper::updateMktDepthL2** will be indicated in the results from the **IBApi::EWrapper↩ ::mktDepthExchanges** field by a 'True' value in the 'isL2' field:

- ```
  public void mktDepthExchanges(DepthMktDataDescription[] depthMktDataDescriptions)
  {
      Console.WriteLine("Market Depth Exchanges:");

      foreach (var depthMktDataDescription in depthMktDataDescriptions)
      {
          Console.WriteLine("Depth Market Data Description: Exchange: {0}, Security Type: {1},
   Listing Exch: {2}, Service Data Type: {3}, Agg Group: {4}",
              depthMktDataDescription.Exchange, depthMktDataDescription.SecType,
              depthMktDataDescription.ListingExch, depthMktDataDescription.ServiceDataType,
              depthMktDataDescription.AggGroup != Int32.MaxValue ? depthMktDataDescription.AggGroup.
  ToString() : ""
              );
      }
  }
  ```

- ```
    @Override
    public void mktDepthExchanges(DepthMktDataDescription[] depthMktDataDescriptions) {
        for (DepthMktDataDescription depthMktDataDescription : depthMktDataDescriptions) {
            System.out.println("Depth Mkt Data Description. Exchange: " + depthMktDataDescription.exchange(
      ) +
            ", ListingExch: " + depthMktDataDescription.listingExch() +
            ", SecType: " + depthMktDataDescription.secType() +
            ", ServiceDataType: " + depthMktDataDescription.serviceDataType() +
            ", AggGroup: " + depthMktDataDescription.aggGroup()
            );
        }
    }
  ```

- ```
     Public Sub mktDepthExchanges(depthMktDataDescriptions As DepthMktDataDescription()) Implements
    EWrapper.mktDepthExchanges
        Console.WriteLine("Market Depth Exchanges:")

        Dim aggGroup As String

        For Each depthMktDataDescription In depthMktDataDescriptions
            If depthMktDataDescription.AggGroup <> Integer.MaxValue Then
                aggGroup = depthMktDataDescription.AggGroup
            Else
                aggGroup = ""
            End If

            Console.WriteLine("Depth Market Data Descriprion. Exchange: " &
     depthMktDataDescription.Exchange &
                              " Security Type: " & depthMktDataDescription.SecType &
                              " Listing Exch: " & depthMktDataDescription.ListingExch &
                              " Service Data Type: " & depthMktDataDescription.ServiceDataType &
                              "  Agg Group: " & aggGroup)
        Next
     End Sub
  ```

- ```
  void TestCppClient::mktDepthExchanges(const std::vector<DepthMktDataDescription> &depthMktDataDescriptions)
      {
      printf("Mkt Depth Exchanges (%lu):\n", depthMktDataDescriptions.size());

      for (unsigned int i = 0; i < depthMktDataDescriptions.size(); i++) {
          printf("Depth Mkt Data Description [%d] - exchange: %s secType: %s listingExch: %s serviceDataType:
     %s aggGroup: %s\n", i,
              depthMktDataDescriptions[i].exchange.c_str(),
              depthMktDataDescriptions[i].secType.c_str(),
              depthMktDataDescriptions[i].listingExch.c_str(),
              depthMktDataDescriptions[i].serviceDataType.c_str(),
              depthMktDataDescriptions[i].aggGroup != INT_MAX ? std::to_string(depthMktDataDescriptions[i].
    aggGroup).c_str() : "");
      }
  }
  ```

- ```
    def mktDepthExchanges(self, depthMktDataDescriptions:ListOfDepthExchanges):
        super().mktDepthExchanges(depthMktDataDescriptions)
        print("mktDepthExchanges:")
        for desc in depthMktDataDescriptions:
            printinstance(desc)
  ```

11.5.3 Receiving

Initially, all requested/available rows will be delivered to the client application. As market moves however these rows will inevitably change. To keep the client's order book consistent, the TWS will send updates not only informing which row is to be updated but also the operation to perform in the row: insert (0), update (1) or remove (2).

- ```
    public class EWrapperImpl : EWrapper
    {

  ...


        public virtual void updateMktDepth(int tickerId, int position, int operation, int side, double
      price, int size)
        {
            Console.WriteLine("UpdateMarketDepth. " + tickerId + " - Position: " + position + ", Operation:
     " + operation + ", Side: " + side + ", Price: " + price + ", Size" + size);
        }


  ...
  ```
  ```

```
        public virtual void updateMktDepthL2(int tickerId, int position, string marketMaker, int operation,
       int side, double price, int size)
        {
            Console.WriteLine("UpdateMarketDepthL2. " + tickerId + " - Position: " + position + ",
       Operation: " + operation + ", Side: " + side + ", Price: " + price + ", Size" + size);
        }
```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void updateMktDepth(int tickerId, int position, int operation,
              int side, double price, int size) {
          System.out.println("UpdateMarketDepth. "+tickerId+" - Position: "+position+", Operation: "+
        operation+", Side: "+side+", Price: "+price+", Size: "+size+"");
      }
  ```

  ...

  ```
      @Override
      public void updateMktDepthL2(int tickerId, int position,
              String marketMaker, int operation, int side, double price, int size) {
          System.out.println("UpdateMarketDepthL2. "+tickerId+" - Position: "+position+", Operation: "+
        operation+", Side: "+side+", Price: "+price+", Size: "+size+"");
      }
  ```

- ```
      Public Class EWrapperImpl
          Implements EWrapper
  ```

  ...

  ```
        Public Sub updateMktDepth(tickerId As Integer, position As Integer, operation As Integer, side As
       Integer, price As Double, size As Integer) Implements IBApi.EWrapper.updateMktDepth
            Console.WriteLine("UpdateMarketDepth. " & CStr(tickerId) & " - Position: " & CStr(position) &
       ", Operation: " & CStr(operation) & ", Side: " & CStr(side) &
                              ", Price: " & CStr(price) & ", Size" & CStr(size))
        End Sub
  ```

  ...

  ```
        Public Sub updateMktDepthL2(tickerId As Integer, position As Integer, marketMaker As String,
       operation As Integer, side As Integer, price As Double, size As Integer) Implements
       IBApi.EWrapper.updateMktDepthL2
            Console.WriteLine("UpdateMarketDepthL2. " & tickerId & " - Position: " & position & ",
       Operation: " & operation & ", Side: " & side &
                              ", Price: " & price & ", Size" & size)
        End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::updateMktDepth(TickerId id, int position, int operation, int side,
                                     double price, int size) {
      printf( "UpdateMarketDepth. %ld - Position: %d, Operation: %d, Side: %d, Price: %g, Size: %d\n", id,
        position, operation, side, price, size);
  }
  ```

  ...

  ```
  void TestCppClient::updateMktDepthL2(TickerId id, int position, const std::string& marketMaker, int
        operation,
                                       int side, double price, int size) {
      printf( "UpdateMarketDepthL2. %ld - Position: %d, Operation: %d, Side: %d, Price: %g, Size: %d\n", id,
        position, operation, side, price, size);
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def updateMktDepth(self, reqId: TickerId, position: int, operation: int,
                         side: int, price: float, size: int):
          super().updateMktDepth(reqId, position, operation, side, price, size)
          print("UpdateMarketDepth. ", reqId, "Position:", position, "Operation:",
                operation, "Side:", side, "Price:", price, "Size", size)
  ```

...

```
    def updateMktDepthL2(self, reqId: TickerId, position: int, marketMaker: str,
                         operation: int, side: int, price: float, size: int):
        super().updateMktDepthL2(reqId, position, marketMaker, operation, side,
                                 price, size)
        print("UpdateMarketDepthL2. ", reqId, "Position:", position, "Operation:",
              operation, "Side:", side, "Price:", price, "Size", size)
```

11.5.4 Canceling

To cancel an active market depth request simply invoke the **IBApi.EClient.cancelMktDepth** passing in the request's identifier.

- ```
      client.cancelMktDepth(2001);
  ```
- ```
      client.cancelMktDepth(2001);
  ```
- ```
      client.cancelMktDepth(2001)
  ```
- ```
  m_pClient->cancelMktDepth(2001);
  ```
- ```
      self.cancelMktDepth(2101)
      self.cancelMktDepth(2001)
  ```

11.5.5 Limitations

Given the potentially high amount of data being sent, market depth request's are much more limited. Just as with historical data requests, the amount of active depth requests is related to the amount of market data lines, with a minimum of three and maximum of 60:

Number of lines	Max. Requests
0 - 399	3
400 - 499	4
500 - 599	5
600 - 699	6
etc...	

11.6 5 Second Real Time Bars

Real time and historical data functionality is combined through the **IBApi.EClient.reqRealTimeBars** request. reqRealTimeBars will create an active subscription that will return a single bar in real time every five seconds that has the OHLC values over that period. reqRealTimeBars can only be used with a bar size of 5 seconds.

Important: real time bars subscriptions combine the limitations of both, top and historical market data. Make sure you observe **Market Data Lines** and **Pacing Violations for Small Bars (30 secs or less)** (p. **??**). For example, no more than 60 ∗**new**∗ requests for real time bars can be made in 10 minutes, and the total number of active active subscriptions of all types cannot exceed the maximum allowed market data lines for the user.
  ```

11.6.1 Requesting

- ```
            client.reqRealTimeBars(3001, ContractSamples.EurGbpFx(), 5, "MIDPOINT", true, null);
  ```
- ```
        client.reqRealTimeBars(3001, ContractSamples.EurGbpFx(), 5, "MIDPOINT", true, null);
  ```
- ```
        client.reqRealTimeBars(3001, ContractSamples.EurGbpFx(), 5, "MIDPOINT", True, Nothing)
  ```
- ```
    m_pClient->reqRealTimeBars(3001, ContractSamples::EurGbpFx(), 5, "MIDPOINT", true, TagValueListSPtr());
  ```
- ```
        self.reqRealTimeBars(3101, ContractSamples.USStockAtSmart(), 5, "MIDPOINT", True, [])
        self.reqRealTimeBars(3001, ContractSamples.EurGbpFx(), 5, "MIDPOINT", True, [])
  ```

- To receive volume, VWAP, or trade count data, it is necessary to specify whatToShow as TRADES.
- It may be necessary to remake real time bars subscriptions after the IB server reset or between trading sessions.

11.6.2 Receiving

Once invoked, historical data bars of five seconds will start being delivered through the **IBApi.EWrapper.realtime↩ Bar** callback:

- ```
    public class EWrapperImpl : EWrapper
    {
  ```

  ...

  ```
        public virtual void realtimeBar(int reqId, long time, double open, double high, double low, double
      close, long volume, double WAP, int count)
        {
            Console.WriteLine("RealTimeBars. " + reqId + " - Time: " + time + ", Open: " + open + ", High:
    " + high + ", Low: " + low + ", Close: " + close + ", Volume: " + volume + ", Count: " + count + ", WAP: " +
     WAP);
        }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
    @Override
    public void realtimeBar(int reqId, long time, double open, double high,
            double low, double close, long volume, double wap, int count) {
        System.out.println("RealTimeBars. " + reqId + " - Time: " + time + ", Open: " + open + ", High: " +
       high + ", Low: " + low + ", Close: " + close + ", Volume: " + volume + ", Count: " + count + ", WAP: " +
      wap);
    }
  ```

- ```
    Public Class EWrapperImpl
        Implements EWrapper
  ```

  ...

  ```
        Public Sub realtimeBar(reqId As Integer, time As Long, open As Double, high As Double, low As
      Double, close As Double, volume As Long, WAP As Double, count As Integer) Implements IBApi.EWrapper.realtimeBar
            Console.WriteLine("RealTimeBars. " & reqId & " - Time: " & time & ", Open: " & open & ", High:
    " & high & ", Low: " & low & ", Close: " &
        close & ", Volume: " & volume & ", Count: " & count & ", WAP: " & WAP)
        End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::realtimeBar(TickerId reqId, long time, double open, double high, double low, double
        close,
                                  long volume, double wap, int count) {
      printf( "RealTimeBars. %ld - Time: %ld, Open: %g, High: %g, Low: %g, Close: %g, Volume: %ld, Count: %d,
        WAP: %g\n", reqId, time, open, high, low, close, volume, count, wap);
  }
  ```
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def realtimeBar(self, reqId:TickerId, time:int, open:float, high:float,
                      low:float, close:float, volume:int, wap:float, count:int):
          super().realtimeBar(reqId, time, open, high, low, close, volume, wap, count)
          print("RealTimeBars. ", reqId, ": time ", time, ", open: ",open,
                ", high: ", high, ", low: ", low, ", close: ", close, ", volume: ", volume,
                ", wap: ", wap, ", count: ", count)
  ```

-Volume for US stocks has a multiplier of 100

11.6.3 Canceling

To cancel an active request simply invoke **IBApi.EClient.cancelRealTimeBars**

- ```
              client.cancelRealTimeBars(3001);
  ```
- ```
          client.cancelRealTimeBars(3001);
  ```
- ```
          client.cancelRealTimeBars(3001)
  ```
- ```
      m_pClient->cancelRealTimeBars(3001);
  ```
- ```
          self.cancelRealTimeBars(3101)
          self.cancelRealTimeBars(3001)
  ```

Chapter 11

Historical Market Data

http://interactivebrokers.github.io/tws-api/historical_data.html

Receiving historical data from the API has the same market data subscription requirement as receiving streaming top-of-book live data **Live Market Data** (p. **??**). The API historical data functionality pulls certain types of data from TWS charts or the historical Time&Sales Window. So if data is not available for a specific instrument, data type, or period within a TWS chart it will also not be available from the API. Unlike TWS, which can create 'delayed charts' for most instruments without any market data subscriptions that have data up until 10-15 minutes prior to the current moment; the API always requires Level 1 streaming real time data to return historical data.

- In general, a **smart-routed** historical data requests will require subscriptions to **all exchanges** on which a instrument trades.
- For instance, a historical data request for a pink sheet (OTC) stock which trades on ARCAEDGE will require the subscription "OTC Global Equities" or "Global OTC Equities and OTC Markets" for ARCAEDGE in addition to the regular subscription (e.g. "OTC Markets").
- When retrieving historical data from the TWS, be aware of the **Historical Data Limitations** (p. **??**).

Types of Historical Data Available

- **Historical Bar Data**
- **Histograms**
- **Historical Time and Sales Data**

Finding earliest date historical data is available for an instrument

- **Finding Earliest Data Point**

Note about Interactive Brokers' historical data:

- Historical data at IB is filtered for trade types which occur away from the NBBO such as combo legs, block trades, and derivative trades. For that reason the daily volume from the (unfiltered) real time data functionality will generally be larger than the (filtered) historical volume reported by historical data functionality. Also, differences are expected in other fields such as the VWAP between the real time and historical data feeds.

12.1 Historical Data Limitations

The maximum number of simultaneous open historical data requests from the API is 50. In practice, it will probably be more efficient to have a much smaller number of requests pending at a time.

12.1.1 Pacing Violations for Small Bars (30 secs or less)

Although Interactive Brokers offers our clients high quality market data, IB is not a specialised market data provider and as such it is forced to put in place restrictions to limit traffic which is not directly associated to trading. A **Pacing Violation**[1] occurs whenever one or more of the following restrictions is not observed:

- Making identical historical data requests within 15 seconds.
- Making six or more historical data requests for the same Contract, Exchange and Tick Type within two seconds.
- Making more than 60 requests within any ten minute period.

*Note that when BID_ASK historical data is requested, each request is counted **twice**. In a nutshell, the information above can simply be put as "do not request too much data too quick".*

Important: the above limitations apply to all our clients and it is not possible to overcome them. If your trading strategy's market data requirements are not met by our market data services please consider contacting a specialised provider.

12.1.2 Step Sizes

A step size is defined as the ratio between the historical data request's duration period and its granularity (i.e. bar size). Historical Data requests need to be assembled in such a way that only a few thousand bars are returned at a time. The following table exemplifies this concept:

Duration	Allowed Bar Sizes
60 S	1 sec - 1 mins
120 S	1 sec - 2 mins
1800 S (30 mins)	1 sec - 30 mins
3600 S (1 hr)	5 secs - 1 hr
14400 S (4hr)	10 secs - 3 hrs
28800 S (8 hrs)	30 secs - 8 hrs
1 D	1 min - 1 day
2 D	2 mins - 1 day
1 W	3 mins - 1 week
1 M	30 mins - 1 month
1 Y	1 day - 1 month

12.1.3 Unavailable Historical Data

- Studies and indicators such as Weighted Moving Averages or Bollinger Bands are not available from the API. VWAP is available.

The other historical data limitations listed are general limitations for all trading platforms:

- Bars which size is 30 seconds or less **older than six months**
- Expired futures data older than two years counting from the future's expiration date.
- Expired options, FOPs, warrants and structured products.
- End of Day (EOD) data for options, FOPs, warrants and structured products.
- Data for expired future spreads
- Data for securities which are no longer trading.
- Native historical data for combos. Historical data is not stored in the IB database separately for combos.; combo historical data in TWS or the API is the sum of data from the legs.
- Historical data for securities which move to a new exchange will often not be available prior to the time of the move.

Note:

1. At this time Historical Data Limitations for barSize = "1 mins" and greater have been lifted. However, please use caution when requesting large amounts of historical data or sending historical data requests too frequently. Though IB has lifted the "hard" limit, we still implement a "soft" slow to load-balance client requests vs. server response. Requesting too much historical data can lead to throttling and eventual disconnect of the API client. If a request requires more than several minutes to return data, it would be best to cancel the request using the ***IBApi.EClient.cancelHistoricalData*** *function.*

12.2 Historical Bar Data

12.2.1 Requesting Historical Bar Data

Historical data is obtained from the the TWS via the **IBApi.EClient.reqHistoricalData** function. Every re-quest needs:

- ***tickerId***, A unique identifier which will serve to identify the incoming data.
- ***contract***, The **IBApi.Contract** you are interested in.
- ***endDateTime***, The request's end date and time (the empty string indicates current present moment).
- ***durationString***, The amount of time (or **Valid Duration String units** (p. **??**)) to go back from the request's given end date and time.
- ***barSizeSetting***, The data's granularity or **Valid Bar Sizes**
- ***whatToShow***, The type of data to retrieve. See **Historical Data Types**
- ***useRTH***, Whether (1) or not (0) to retrieve data generated only within Regular Trading Hours (RTH)
- ***formatDate***, The format in which the incoming bars' date should be presented. Note that for day bars, only yyyyMMdd format is available.

- ***keepUpToDate***, Whether a subscription is made to return updates of unfinished real time bars as they are available (True), or all data is returned on a one-time basis (False). *Available starting with API v973.03+ and TWS v965+*. If *True*, and endDateTime cannot be specified.

For example, making a request with an end date and time of "20160127 23:59:59", a duration string of "3 D" and a bar size of "1 hour" will return three days worth of 1 hour bars data in which the most recent bar will be the closest possible to 20160127 23:59:59.

```
            String queryTime = DateTime.Now.AddMonths(-6).ToString("yyyyMMdd HH:mm:ss");
            client.reqHistoricalData(4001, ContractSamples.EurGbpFx(), queryTime, "1 M", "1 day", "MIDPOINT
        ", 1, 1, false, null);
            client.reqHistoricalData(4002, ContractSamples.EuropeanStock(), queryTime, "10 D", "1 min", "
        TRADES", 1, 1, false, null);
```

```
          Calendar cal = Calendar.getInstance();
          cal.add(Calendar.MONTH, -6);
          SimpleDateFormat form = new SimpleDateFormat("yyyyMMdd HH:mm:ss");
          String formatted = form.format(cal.getTime());
          client.reqHistoricalData(4001, ContractSamples.EurGbpFx(), formatted, "1 M", "1 day", "MIDPOINT", 1
        , 1, false, null);
          client.reqHistoricalData(4002, ContractSamples.EuropeanStock(), formatted, "10 D", "1 min", "TRADES
        ", 1, 1, false, null);
          Thread.sleep(2000);
          /*** Canceling historical data requests ***/
          client.cancelHistoricalData(4001);
          client.cancelHistoricalData(4002);
```

```
          Dim queryTime As String = DateTime.Now.AddMonths(-6).ToString("yyyyMMdd HH:mm:ss")
          client.reqHistoricalData(4001, ContractSamples.EurGbpFx(), queryTime, "1 M", "1 day", "MIDPOINT",
         1, 1, False, Nothing)
          client.reqHistoricalData(4002, ContractSamples.EuropeanStock(), queryTime, "10 D", "1 min",
        "TRADES", 1, 1, False, Nothing)
```

```
    std::time_t rawtime;
    std::tm* timeinfo;
    char queryTime [80];

    std::time(&rawtime);
    timeinfo = std::localtime(&rawtime);
    std::strftime(queryTime, 80, "%Y%m%d %H:%M:%S", timeinfo);

    m_pClient->reqHistoricalData(4001, ContractSamples::EurGbpFx(), queryTime, "1 M", "1 day", "MIDPOINT",
      1, 1, false, TagValueListSPtr());
    m_pClient->reqHistoricalData(4002, ContractSamples::EuropeanStock(), queryTime, "10 D", "1 min", "
      TRADES", 1, 1, false, TagValueListSPtr());
```

```
        queryTime = (datetime.datetime.today() -
                     datetime.timedelta(days=180)).strftime("%Y%m%d %H:%M:%S")
        self.reqHistoricalData(4101, ContractSamples.USStockAtSmart(), queryTime,
                               "1 M", "1 day", "MIDPOINT", 1, 1, False, [])
        self.reqHistoricalData(4001, ContractSamples.EurGbpFx(), queryTime,
                               "1 M", "1 day", "MIDPOINT", 1, 1, False, [])
        self.reqHistoricalData(4002, ContractSamples.EuropeanStock(), queryTime,
                               "10 D", "1 min", "TRADES", 1, 1, False, [])
```

The daily bar size has several unique characteristics. This is true both in TWS and the API:

- For futures, the close price of daily bars can be the *settlement* price if provided by the exchange. Generally the official settlement price is not available until several hours after a trading session closes. The Friday settlement price will sometimes not be available until Saturday.
- A daily bar will refer to a trading session which may cross calendar days. In that case the date of the bar will correspond to the day on which the bar closes.

12.2.2 Receiving Historical Bar Data

The historical data will be delivered via the **IBApi::EWrapper::historicalData** method in the form of **candle-sticks**. The time zone of returned bars is the time zone chosen in TWS on the login screen. If reqHistoricalData was invoked with ***keepUpToDate = false***, once all candlesticks have been received the **IBApi.EWrapper.historical↔DataEnd** marker will be sent. Otherwise updates of the most recent partial five-second bar will continue to be returned in real time to **IBApi::EWrapper::historicalDataUpdate** (p. **??**). The keepUpToDate functionality can only be used with bar sizes 5 seconds or greater and requires the endDate is set as the empty string.

- Note: IB's historical data feed is filtered for some types of trades which generally occur away from the NBBO such as combos, block trades, and derivatives. For that reason the historical data volume will be lower than an unfiltered historical data feed.
- IB does not maintain separate historical data for combos. Historical data returned for a combo contract will be the sum of data from the individual legs.

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
      public virtual void historicalData(int reqId, Bar bar)
      {
          Console.WriteLine("HistoricalData. " + reqId + " - Time: " + bar.Time + ", Open: " + bar.Open +
     ", High: " + bar.High + ", Low: " + bar.Low + ", Close: " + bar.Close + ", Volume: " + bar.Volume + ",
     Count: " + bar.Count + ", WAP: " + bar.WAP);
      }
  ```

  ...

  ```
      public virtual void historicalDataEnd(int reqId, string startDate, string endDate)
      {
          Console.WriteLine("HistoricalDataEnd - "+reqId+" from "+startDate+" to "+endDate);
      }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
  @Override
  public void historicalData(int reqId, Bar bar) {
      System.out.println("HistoricalData. "+reqId+" - Date: "+bar.time()+", Open: "+bar.open()+", High: "
    +bar.high()+", Low: "+bar.low()+", Close: "+bar.close()+", Volume: "+bar.volume()+", Count: "+bar.count()+",
     WAP: "+bar.wap());
  }
  ```

  ...

  ```
  @Override
  public void historicalDataEnd(int reqId, String startDateStr, String endDateStr) {
      System.out.println("HistoricalDataEnd. "+reqId+" - Start Date: "+startDateStr+", End Date: "+
    endDateStr);
  }
  ```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

  ...

  ```
      Public Sub historicalData(reqId As Integer, bar As Bar) Implements IBApi.EWrapper.historicalData
          Console.WriteLine("HistoricalData - ReqId [" & reqId & "] Date [" & bar.Time & "] Open [" &
     bar.Open & "] High [" &
                            bar.High & "] Low [" & bar.Low & "] Volume [" & bar.Volume & "] Count [" &
     bar.Count & "]")
      End Sub
  ```

  ...

  ```
      Public Sub historicalDataEnd(reqId As Integer, start As String, [end] As String) Implements
     IBApi.EWrapper.historicalDataEnd
          Console.WriteLine("HistoricalDataEnd - ReqId [" & reqId & "] Start [" & start & "] End [" &
     [end] & "]")
      End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::historicalData(TickerId reqId, const Bar& bar) {
      printf( "HistoricalData. ReqId: %ld - Date: %s, Open: %g, High: %g, Low: %g, Close: %g, Volume: %lld,
        Count: %d, WAP: %g\n", reqId, bar.time.c_str(), bar.open, bar.high, bar.low, bar.close, bar.volume, bar.count
        , bar.wap);
  }
  ```

...

```
void TestCppClient::historicalDataEnd(int reqId, const std::string& startDateStr, const std::string&
      endDateStr) {
    std::cout << "HistoricalDataEnd. ReqId: " << reqId << " - Start Date: " << startDateStr << ", End Date:
       " << endDateStr << std::endl;
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

...

```
    def historicalData(self, reqId:int, bar: BarData):
        print("HistoricalData. ", reqId, " Date:", bar.date, "Open:", bar.open,
              "High:", bar.high, "Low:", bar.low, "Close:", bar.close, "Volume:", bar.volume,
              "Count:", bar.barCount, "WAP:", bar.average)
```

...

```
    def historicalDataEnd(self, reqId: int, start: str, end: str):
        super().historicalDataEnd(reqId, start, end)
        print("HistoricalDataEnd ", reqId, "from", start, "to", end)
```

- ```
          public void historicalDataUpdate(int reqId, Bar bar)
          {
              Console.WriteLine("HistoricalDataUpdate. " + reqId + " - Time: " + bar.Time + ", Open: " + bar.
        Open + ", High: " + bar.High + ", Low: " + bar.Low + ", Close: " + bar.Close + ", Volume: " + bar.Volume + "
        , Count: " + bar.Count + ", WAP: " + bar.WAP);
          }
  ```

- ```
      @Override
      public void historicalDataUpdate(int reqId, Bar bar) {
          System.out.println("HistoricalDataUpdate. "+reqId+" - Date: "+bar.time()+", Open: "+bar.open()+",
         High: "+bar.high()+", Low: "+bar.low()+", Close: "+bar.close()+", Volume: "+bar.volume()+", Count: "+bar.
        count()+", WAP: "+bar.wap());
      }
  ```

- ```
          Public Sub historicalDataUpdate(reqId As Integer, bar As Bar) Implements
         IBApi.EWrapper.historicalDataUpdate
              Console.WriteLine("HistoricalDataUpdate - ReqId [" & reqId & "] Date [" & bar.Time & "] Open ["
        & bar.Open & "] High [" &
                          bar.High & "] Low [" & bar.Low & "] Volume [" & bar.Volume & "] Count [" &
        bar.Count & "]")
          End Sub
  ```

- ```
  void TestCppClient::historicalDataUpdate(TickerId reqId, const Bar& bar) {
      printf( "HistoricalDataUpdate. ReqId: %ld - Date: %s, Open: %g, High: %g, Low: %g, Close: %g, Volume:
         %lld, Count: %d, WAP: %g\n", reqId, bar.time.c_str(), bar.open, bar.high, bar.low, bar.close, bar.volume, bar
        .count, bar.wap);
  }
  ```

- ```
      def historicalDataUpdate(self, reqId: int, bar: BarData):
          print("HistoricalDataUpdate. ", reqId, " Date:", bar.date, "Open:", bar.open,
                "High:", bar.high, "Low:", bar.low, "Close:", bar.close, "Volume:", bar.volume,
                "Count:", bar.barCount, "WAP:", bar.average)
  ```

12.2.3 Valid Duration String units

Unit	Description
S	Seconds
D	Day
W	Week
M	Month
Y	Year

12.2.4 Valid Bar Sizes

	Size						
1 secs	5 secs	10 secs	15 secs	30 secs			
1 min	2 mins	3 mins	5 mins	10 mins	15 mins	20 mins	30 mins
1 hour	2 hours	3 hours	4 hours	8 hours			
1 day							
1 week							
1 month							

12.2.5 Historical Data Types

(whatToShow)

All different kinds of historical data are returned in the form of candlesticks and as such the values return represent the state of the market **during the period covered by the candlestick**. For instance, it might appear obvious but a TRADES day bar's 'close' value does **NOT** represent the product's closing price but rather the last traded price registered.

Type	Open	High	Low	Close	Volume
TRADES	First traded price	Highest traded price	Lowest traded price	Last traded price	Total traded volume
MIDPOINT	Starting midpoint price	Highest midpoint price	Lowest midpoint price	Last midpoint price	N/A
BID	Starting bid price	Highest bid price	Lowest bid price	Last bid price	N/A
ASK	Starting ask price	Highest ask price	Lowest ask price	Last ask price	N/A
BID_ASK	Time average bid	Max Ask	Min Bid	Time average ask	N/A
ADJUSTED_LAST	Dividend-adjusted first traded price	Dividend-adjusted high trade	Dividend-adjusted low trade	Dividend-adjusted last trade	Total traded volume
HISTORICAL_VOLATILITY	Starting volatility	Highest volatility	Lowest volatility	Last volatility	N/A
OPTION_IMPLIED_VOLATILITY	Starting implied volatility	Highest implied volatility	Lowest implied volatility	Last implied volatility	N/A
REBATE_RATE	Starting rebate rate	Highest rebate rate	Lowest rebate rate	Last rebate rate	N/A
FEE_RATE	Starting fee rate	Highest fee rate	Lowest fee rate	Last fee rate	N/A
YIELD_BID	Starting bid yield	Highest bid yield	Lowest bid yield	Last bid yield	N/A
YIELD_ASK	Starting ask yield	Highest ask yield	Lowest ask yield	Last ask yield	N/A
YIELD_BID_ASK	Time average bid yield	Highest ask yield	Lowest bid yield	Time average ask yield	N/A
YIELD_LAST	Starting last yield	Highest last yield	Lowest last yield	Last last yield	N/A

- TRADES data is adjusted for splits, but not dividends
- ADJUSTED_LAST data is adjusted for splits and dividends. Requires TWS 967+.

12.2.6 Available Data per Product

Product Type	TRADES	MIDPOINT	BID	ASK	BID_ASK	HISTORICAL_VOLATILITY	OPTION_IMPLIED_VOLATILITY	YIELD_BID	YIELD_ASK	YIELD_BID_ASK	YIELD_LAST
Stocks	Y	Y	Y	Y	Y	Y	Y	N	N	N	N
Commodities	N	Y	Y	Y	Y	N	N	N	N	N	N
Options	Y	Y	Y	Y	Y	N	N	N	N	N	N
Futures	Y	Y	Y	Y	Y	N	N	N	N	N	N
FOPs	Y	Y	Y	Y	Y	N	N	N	N	N	N
ETFs	Y	Y	Y	Y	Y	Y	Y	N	N	N	N
Warrants	Y	Y	Y	Y	Y	N	N	N	N	N	N
Structured Products	Y	Y	Y	Y	Y	N	N	N	N	N	N
SSFs	Y	Y	Y	Y	Y	N	N	N	N	N	N
Forex	N	Y	Y	Y	Y	N	N	N	N	N	N
Metals	Y	Y	Y	Y	Y	N	N	N	N	N	N

Product Type	TRADES	MIDPOINT	BID	ASK	BID_ASK	HISTORICAL_VOLATILITY	OPTION_IMPLIED_VOLATILITY	YIELD_BID	YIELD_ASK	YIELD_BID_ASK	YIELD_LAST
Indices	Y	N	N	N	N	Y	Y	N	N	N	N
Bonds*	Y	Y	Y	Y	Y	N	N	Y	Y	Y	Y
Funds	N	Y	Y	Y	Y	N	N	N	N	N	N
CFDs*	N	Y	Y	Y	Y	N	N	N	N	N	N

-Yield historical data only available for corporate bonds

12.3 Histograms

Histogram data requires API v973.02+ and TWS 964+. Instead of returned data points as a function of time as with the function **IBApi::EClient::reqHistoricalData** (p. **??**), histograms return data as a function of price level with function **IBApi::EClient::reqHistogramData**

- `client.reqHistogramData(15001, ContractSamples.USStockWithPrimaryExch(), false, "1 week");`
- `/*client.reqHistogramData(4004, ContractSamples.USStock(), false, "3 days");`
- `client.reqHistogramData(15001, ContractSamples.USStockWithPrimaryExch, False, "1 week")`
- `m_pClient->reqHistogramData(15001, ContractSamples::IBMUSStockAtSmart(), false, "1 weeks");`
- `self.reqHistogramData(4104, ContractSamples.USStock(), False, "3 days")`

and data is returned to **IBApi::EWrapper::histogramData**

- ```
  public void histogramData(int reqId, HistogramEntry[] data)
  {
      Console.WriteLine("Histogram data. Request Id: {0}, data size: {1}", reqId, data.Length);
      data.ToList().ForEach(i => Console.WriteLine("\tPrice: {0}, Size: {1}", i.Price, i.Size));
  }
  ```
  ```

```
    @Override
    public void histogramData(int reqId, List<HistogramEntry> items) {
        System.out.println(EWrapperMsgGenerator.histogramData(reqId, items));
    }
```

```
        Public Sub histogramData(reqId As Integer, data As HistogramEntry()) Implements
       EWrapper.histogramData
            Console.WriteLine("Histogram data. Request Id: {0}, data size: {1}", reqId, data.Length)
            data.ToList().ForEach(Sub(i) Console.WriteLine(vbTab & "Price: {0}, Size: {1}", i.Price,
       i.Size))
        End Sub
```

```
void TestCppClient::histogramData(int reqId, const HistogramDataVector& data) {
    printf("Histogram. ReqId: %d, data length: %lu\n", reqId, data.size());

    for (auto item : data) {
        printf("\t price: %f, size: %lld\n", item.price, item.size);
    }
}
```

```
    def histogramData(self, reqId:int, items:HistogramDataList):
        print("HistogramData: ", reqId, " ", items)
```

An active histogram request which has not returned data can be cancelled with **IBApi::EClient::cancel↩ HistogramData**

```
            client.cancelHistogramData(15001);
```

```
        client.cancelHistogramData(4004);*/
```

```
        client.cancelHistogramData(15001)
```

```
    m_pClient->cancelHistogramData(15001);
```

```
        self.cancelHistogramData(4104)
```

12.4 Historical Time and Sales Data

12.4.1 High Resolution Historical Data

The highest granularity of historical data from IB's database can be retrieved using the API function **IBApi::↩ EClient::reqHistoricalTicks** (p. **??**). This corresponds to the `TWS Historical Time & Sales` Window. TWS build **968+** and API version **973.04+** is required.

- Historical Tick-By-Tick data is not available for combos

- **Note:** the historical Time&Sales feature in Python API is available starting in API v973.06+.

```
client.reqHistoricalTicks(18001, ContractSamples.USStockAtSmart(), "20170712 21:39:33", null,
10, "TRADES", 1, true, null);
client.reqHistoricalTicks(18002, ContractSamples.USStockAtSmart(), "20170712 21:39:33", null,
10, "BID_ASK", 1, true, null);
client.reqHistoricalTicks(18003, ContractSamples.USStockAtSmart(), "20170712 21:39:33", null,
10, "MIDPOINT", 1, true, null);
```

```
client.reqHistoricalTicks(18001, ContractSamples.USStockAtSmart(), "20170712 21:39:33", null, 10, "
TRADES", 1, true, null);
client.reqHistoricalTicks(18002, ContractSamples.USStockAtSmart(), "20170712 21:39:33", null, 10, "
BID_ASK", 1, true, null);
client.reqHistoricalTicks(18003, ContractSamples.USStockAtSmart(), "20170712 21:39:33", null, 10, "
MIDPOINT", 1, true, null);
```

```
client.reqHistoricalTicks(18001, ContractSamples.USStockAtSmart(), "20170712 21:39:33", Nothing,
 10, "TRADES", 1, True, Nothing)
client.reqHistoricalTicks(18002, ContractSamples.USStockAtSmart(), "20170712 21:39:33", Nothing,
 10, "BID_ASK", 1, True, Nothing)
client.reqHistoricalTicks(18003, ContractSamples.USStockAtSmart(), "20170712 21:39:33", Nothing,
 10, "MIDPOINT", 1, True, Nothing)
```

```
m_pClient->reqHistoricalTicks(19001, ContractSamples::IBMUSStockAtSmart(), "20170621 09:38:33", "", 10,
    "BID_ASK", 1, true, TagValueListSPtr());
m_pClient->reqHistoricalTicks(19002, ContractSamples::IBMUSStockAtSmart(), "20170621 09:38:33", "", 10,
    "MIDPOINT", 1, true, TagValueListSPtr());
m_pClient->reqHistoricalTicks(19003, ContractSamples::IBMUSStockAtSmart(), "20170621 09:38:33", "", 10,
    "TRADES", 1, true, TagValueListSPtr());
```

```
self.reqHistoricalTicks(18001, ContractSamples.USStockAtSmart(),
                        "20170712 21:39:33", "", 10, "TRADES", 1, True, [])
self.reqHistoricalTicks(18002, ContractSamples.USStockAtSmart(),
                        "20170712 21:39:33", "", 10, "BID_ASK", 1, True, [])
self.reqHistoricalTicks(18003, ContractSamples.USStockAtSmart(),
                        "20170712 21:39:33", "", 10, "MIDPOINT", 1, True, [])
```

- ***requestId***, id of the request
- ***contract***, Contract object that is subject of query
- ***startDateTime***, i.e. "20170701 12:01:00". Uses TWS timezone specified at login.
- ***endDateTime***, i.e. "20170701 13:01:00". In TWS timezone. Exactly one of start time and end time has to be defined.
- ***numberOfTicks***, Number of distinct data points. Max currently 1000 per request.
- ***whatToShow***, (Bid_Ask, Midpoint, Trades) Type of data requested.
- ***useRth***, Data from regular trading hours (1), or all available hours (0)
- ***ignoreSize***, A filter only used when the source price is Bid_Ask
- ***miscOptions*** should be defined as *null*, reserved for internal use

Data is returned to the functions **IBApi.EWrapper.historicalTicks** (p. **??**), **IBApi.EWrapper.historicalTicksBid↩Ask** (p. **??**), and **IBApi.EWrapper.historicalTicksLast** (p. **??**), depending on the type of data requested.

```
public void historicalTicks(int reqId, HistoricalTick[] ticks, bool done)
{
    foreach (var tick in ticks)
    {
        Console.WriteLine("Historical Tick. Request Id: {0}, Time: {1}, Price: {2}, Size: {3}",
reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Price, tick.Size);
    }
}
```

...

```
public void historicalTicksBidAsk(int reqId, HistoricalTickBidAsk[] ticks, bool done)
{
    foreach (var tick in ticks)
    {
        Console.WriteLine("Historical Tick Bid/Ask. Request Id: {0}, Time: {1}, Mask: {2} Price
 Bid: {3}, Price Ask {4}, Size Bid: {5}, Size Ask {6}",
            reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Mask, tick.
PriceBid, tick.PriceAsk, tick.SizeBid, tick.SizeAsk);
    }
}
```

...

```
        public void historicalTicksLast(int reqId, HistoricalTickLast[] ticks, bool done)
        {
            foreach (var tick in ticks)
            {
                Console.WriteLine("Historical Tick Last. Request Id: {0}, Time: {1}, Mask: {2}, Price: {3},
    Size: {4}, Exchange: {5}, Special Conditions: {6}",
                    reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Mask, tick.
  Price, tick.Size, tick.Exchange, tick.SpecialConditions);
            }
        }
```

- ```
    @Override
    public void historicalTicks(int reqId, List<HistoricalTick> ticks, boolean done) {
        for (HistoricalTick tick : ticks) {
            System.out.println(EWrapperMsgGenerator.historicalTick(reqId, tick.time(), tick.price(), tick.
  size()));
        }
    }
```

...

```
    @Override
    public void historicalTicksBidAsk(int reqId, List<HistoricalTickBidAsk> ticks, boolean done) {
        for (HistoricalTickBidAsk tick : ticks) {
            System.out.println(EWrapperMsgGenerator.historicalTickBidAsk(reqId, tick.time(), tick.mask(),
  tick.priceBid(), tick.priceAsk(), tick.sizeBid(),
                    tick.sizeAsk()));
        }
    }
```

...

```
    public void historicalTicksLast(int reqId, List<HistoricalTickLast> ticks, boolean done) {
        for (HistoricalTickLast tick : ticks) {
            System.out.println(EWrapperMsgGenerator.historicalTickLast(reqId, tick.time(), tick.mask(),
  tick.price(), tick.size(), tick.exchange(),
                    tick.specialConditions()));
        }
    }
```

- ```
        Public Sub historicalTick(reqId As Integer, ticks As HistoricalTick(), done As Boolean) Implements
    EWrapper.historicalTicks
            For Each tick In ticks
                Console.WriteLine("Historical Tick. Request Id: {0}, Time: {1}, Price: {2}, Size: {3}",
    reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Price, tick.Size)
            Next
        End Sub
```

...

```
        Public Sub historicalTickBidAsk(reqId As Integer, ticks As HistoricalTickBidAsk(), done As Boolean)
    Implements EWrapper.historicalTicksBidAsk
            For Each tick In ticks
                Console.WriteLine("Historical Tick Bid/Ask. Request Id: {0}, Time: {1}, Mask: {2} Price
    Bid: {3}, Price Ask {4}, Size Bid: {5}, Size Ask {6}",
                    reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Mask,
    tick.PriceBid, tick.PriceAsk, tick.SizeBid, tick.SizeAsk)
            Next
        End Sub
```

...

```
        Public Sub historicalTickLast(reqId As Integer, ticks As HistoricalTickLast(), done As Boolean)
    Implements EWrapper.historicalTicksLast
            For Each tick In ticks
                Console.WriteLine("Historical Tick Last. Request Id: {0}, Time: {1}, Mask: {2}, Price: {3},
    Size: {4}, Exchange: {5}, Special Conditions: {6}",
                    reqId, Util.UnixSecondsToString(tick.Time, "yyyyMMdd-HH:mm:ss zzz"), tick.Mask,
    tick.Price, tick.Size, tick.Exchange, tick.SpecialConditions)
            Next
        End Sub
```

- ```
  void TestCppClient::historicalTicks(int reqId, const std::vector<HistoricalTick>& ticks, bool done) {
      for (HistoricalTick tick : ticks) {
      std::time_t t = tick.time;
          std::cout << "Historical tick. ReqId: " << reqId << ", time: " << ctime(&t) << ", price: "<< tick.
    price << ", size: " << tick.size << std::endl;
      }
  }
  ```

...
  ```

```
void TestCppClient::historicalTicksBidAsk(int reqId, const std::vector<HistoricalTickBidAsk>& ticks, bool
      done) {
    for (HistoricalTickBidAsk tick : ticks) {
    std::time_t t = tick.time;
        std::cout << "Historical tick bid/ask. ReqId: " << reqId << ", time: " << ctime(&t) << ", mask: " <
    < tick.mask << ", price bid: "<< tick.priceBid <<
          ", price ask: "<< tick.priceAsk << ", size bid: " << tick.sizeBid << ", size ask: " << tick.
    sizeAsk << std::endl;
    }
}
```

...

```
void TestCppClient::historicalTicksLast(int reqId, const std::vector<HistoricalTickLast>& ticks, bool done)
      {
    for (HistoricalTickLast tick : ticks) {
    std::time_t t = tick.time;
        std::cout << "Historical tick last. ReqId: " << reqId << ", time: " << ctime(&t) << ", mask: " <<
    tick.mask << ", price: "<< tick.price <<
          ", size: " << tick.size << ", exchange: " << tick.exchange << ", special conditions: " << tick.
    specialConditions << std::endl;
    }
}
```

-
```
    def historicalTicks(self, reqId: int, ticks: ListOfHistoricalTick, done: bool):
        for tick in ticks:
            print("Historical Tick. Req Id: ", reqId, ", time: ", tick.time,
                  ", price: ", tick.price, ", size: ", tick.size)
```

...

```
    def historicalTicksBidAsk(self, reqId: int, ticks: ListOfHistoricalTickBidAsk,
                              done: bool):
        for tick in ticks:
            print("Historical Tick Bid/Ask. Req Id: ", reqId, ", time: ", tick.time,
                  ", bid price: ", tick.priceBid, ", ask price: ", tick.priceAsk,
                  ", bid size: ", tick.sizeBid, ", ask size: ", tick.sizeAsk)
```

...

```
    def historicalTicksLast(self, reqId: int, ticks: ListOfHistoricalTickLast,
                            done: bool):
        for tick in ticks:
            print("Historical Tick Last. Req Id: ", reqId, ", time: ", tick.time,
                  ", price: ", tick.price, ", size: ", tick.size, ", exchange: ", tick.exchange,
                  ", special conditions:", tick.specialConditions)
```

12.5 Finding Earliest Data Point

12.5.1 Earliest Available Data

To find the earliest available data point for a given instrument and data type a function is in the API starting in v973.02 and v963 of TWS/IBG, **IBApi::EClient::reqHeadTimestamp**

-
```
            client.reqHeadTimestamp(14001, ContractSamples.USStock(), "TRADES", 1, 1);
```

-
```
        client.reqHeadTimestamp(4003, ContractSamples.USStock(), "TRADES", 1, 1);
```

-
```
        client.reqHeadTimestamp(14001, ContractSamples.USStock(), "TRADES", 1, 1)
```

-
```
    m_pClient->reqHeadTimestamp(14001, ContractSamples::EurGbpFx(), "MIDPOINT", 1, 1);
```

-
```
        self.reqHeadTimeStamp(4103, ContractSamples.USStockAtSmart(), "TRADES", 0, 1)
```

The resulting head timestamp is returned to the function **IBApi.EWrapper.headTimestamp**

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
          public void headTimestamp(int reqId, string headTimestamp)
          {
              Console.WriteLine("Head time stamp. Request Id: {0}, Head time stamp: {1}", reqId,
        headTimestamp);
          }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void headTimestamp(int reqId, String headTimestamp) {
          System.out.println("Head timestamp. Req Id: " + reqId + ", headTimestamp: " + headTimestamp);
      }
  ```

- ```
      Public Class EWrapperImpl
          Implements EWrapper
  ```

  ...

  ```
          Public Sub headTimestamp(requestId As Integer, timeStamp As String) Implements
         IBApi.EWrapper.headTimestamp
              Console.WriteLine("Head time stamp. Request Id: {0}, Head time stamp: {1}", requestId,
         timeStamp)
          End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::headTimestamp(int reqId, const std::string& headTimestamp) {
      printf( "Head time stamp. ReqId: %d - Head time stamp: %s,\n", reqId, headTimestamp.c_str());

  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def headTimestamp(self, reqId:int, headTimestamp:str):
          print("HeadTimestamp: ", reqId, " ", headTimestamp)
  ```

A reqHeadTimeStamp request can be cancelled with **IBApi.EClient.cancelHeadTimestamp**

- ```
              client.cancelHeadTimestamp(14001);
  ```

- ```
          client.cancelHeadTimestamp(4003);
  ```

- ```
          client.cancelHeadTimestamp(14001)
  ```

- ```
      m_pClient->cancelHeadTimestamp(14001);
  ```

- ```
          self.cancelHeadTimeStamp(4103)
  ```

Chapter 12

Account & Portfolio Data

http://interactivebrokers.github.io/tws-api/account_portfolio.html

The TWS offers a comprehensive overview of your account and portfolio through its Account and Portfolio windows. This information can be obtained via the TWS API through three different kind of requests/operations:

- **Managed Accounts**
- **Family Codes**
- **Account Updates**
- **Account Summary**
- **Positions**
- **Profit And Loss (P&L)**

13.1 Managed Accounts

A single user name can handle more than one account. As mentioned in the **Connectivity** section, the TWS will automatically send a list of managed accounts once the connection is established. The list can also be fetched via the **IBApi.EClient.reqManagedAccts** method:

```
            client.reqManagedAccts();
```

```
        client.reqManagedAccts();
```

```
        client.reqManagedAccts()
```

```
    m_pClient->reqManagedAccts();
```

```
        self.reqManagedAccts()
```

The above will result in a comma separated list of account Ids delivered through **IBApi.EWrapper.managed↩ Accounts**

```
    public class EWrapperImpl : EWrapper
    {
```

...

```
        public virtual void managedAccounts(string accountsList)
        {
            Console.WriteLine("Account list: "+accountsList);
        }
```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void managedAccounts(String accountsList) {
          System.out.println("Account list: " +accountsList);
      }
  ```

- ```
      Public Class EWrapperImpl
          Implements EWrapper
  ```

  ...

  ```
          Public Sub managedAccounts(accountsList As String) Implements IBApi.EWrapper.managedAccounts
              Console.WriteLine("ManagedAccounts - AccountsList [" & accountsList & "]")
          End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::managedAccounts( const std::string& accountsList) {
      printf( "Account List: %s\n", accountsList.c_str());
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def managedAccounts(self, accountsList: str):
          super().managedAccounts(accountsList)
          print("Account list: ", accountsList)
  ```

Important: whenever your TWS user name handles more than a single account, you will be forced to specify the account Id to which the order needs to be allocated. Failure to do so will result in the order being rejected since the TWS cannot assign the order to a default account.

13.2 Family Codes

Starting with API **v973.02** and TWS **v964**, it is possible to determine from the API whether an account exists under an account family, and find the family code using the function reqFamilyCodes.

- ```
              client.reqFamilyCodes();
  ```
- ```
          client.reqFamilyCodes();
  ```
- ```
          client.reqFamilyCodes()
  ```
- ```
      m_pClient->reqFamilyCodes();
  ```
- ```
          self.reqFamilyCodes()
  ```

If a managed account is part of an account family (for instance an individual account under a Financial Advisor or IBroker), that family will be returned to the function **IBApi::EWrapper** (p. **??**):familyCodes. Otherwise it will return an empty string.
  ```

```
        public void familyCodes(FamilyCode[] familyCodes)
        {
            Console.WriteLine("Family Codes:");

            foreach (var familyCode in familyCodes)
            {
                Console.WriteLine("Account ID: {0}, Family Code Str: {1}", familyCode.AccountID, familyCode
    .FamilyCodeStr);
            }
        }
```

```
    @Override
    public void familyCodes(FamilyCode[] familyCodes) {
        for (FamilyCode fc : familyCodes) {
            System.out.print("Family Code. AccountID: " + fc.accountID() + ", FamilyCode: " + fc.
  familyCodeStr());
        }

        System.out.println();
    }
```

```
        Public Sub familyCodes(familyCodes As FamilyCode()) Implements EWrapper.familyCodes
            Console.WriteLine("Family Codes:")

            For Each familyCode In familyCodes
                Console.WriteLine("Account ID: " & familyCode.AccountID & " Family Code Str: " &
     familyCode.FamilyCodeStr)
            Next
        End Sub
```

```
void TestCppClient::familyCodes(const std::vector<FamilyCode> &familyCodes) {
    printf("Family codes (%lu):\n", familyCodes.size());

    for (unsigned int i = 0; i < familyCodes.size(); i++) {
        printf("Family code [%d] - accountID: %s familyCodeStr: %s\n", i, familyCodes[i].accountID.c_str(),
       familyCodes[i].familyCodeStr.c_str());
    }
}
```

```
    def familyCodes(self, familyCodes: ListOfFamilyCode):
        super().familyCodes(familyCodes)
        print("Family Codes:")
        for familyCode in familyCodes:
            print("Account ID: %s, Family Code Str: %s" % (
                familyCode.accountID, familyCode.familyCodeStr))
```

For instance, if individual account U112233 is under a financial advisor with account number F445566, if the function reqFamilyCodes is invoked for the user of account U112233, the family code "F445566A" will be returned, indicating that it belongs within that account family.

13.3 Account Updates

13.3.1 Requesting

The **IBApi.EClient.reqAccountUpdates** function creates a subscription to the TWS through which account and portfolio information is delivered. This information is the exact same as the one displayed within the TWS' Account Window. Note this function receives a specific account along with a flag indicating whether to start or stop the subscription. In a single account structure, the account number is not necessary. Just as with the TWS' Account Window, unless there is a position change this information is updated at a fixed interval of **three** minutes.

```
            client.reqAccountUpdates(true, "U150462");
```

```
        client.reqAccountUpdates(true, "U150462");
```

```
        client.reqAccountUpdates(True, "U150462")
```

```
    m_pClient->reqAccountUpdates(true, "U150462");
```

```
        self.reqAccountUpdates(True, self.account)
```

13.3.2 Receiving

Resulting account and portfolio information will be delivered via the **IBApi.EWrapper.updateAccountValue** (p. **??**), **IBApi.EWrapper.updatePortfolio** (p. **??**), **IBApi.EWrapper.updateAccountTime** and **IBApi.EWrapper.accountDownloadEnd**

-
```
public class EWrapperImpl : EWrapper
{
```

...

```
  public virtual void updateAccountValue(string key, string value, string currency, string
accountName)
  {
      Console.WriteLine("UpdateAccountValue. Key: " + key + ", Value: " + value + ", Currency: " +
currency + ", AccountName: " + accountName);
  }
```

...

```
  public virtual void updatePortfolio(Contract contract, double position, double marketPrice, double
marketValue, double averageCost, double unrealizedPNL, double realizedPNL, string accountName)
  {
      Console.WriteLine("UpdatePortfolio. "+contract.Symbol+", "+contract.SecType+" @ "+contract.
Exchange
          +": Position: "+position+", MarketPrice: "+marketPrice+", MarketValue: "+marketValue+",
 AverageCost: "+averageCost
          +", UnrealizedPNL: "+unrealizedPNL+", RealizedPNL: "+realizedPNL+", AccountName: "+
accountName);
  }
```

...

```
  public virtual void updateAccountTime(string timestamp)
  {
      Console.WriteLine("UpdateAccountTime. Time: " + timestamp+"\n");
  }
```

...

```
  public virtual void accountDownloadEnd(string account)
  {
      Console.WriteLine("Account download finished: "+account+"\n");
  }
```

-
```
public class EWrapperImpl implements EWrapper {
```

...

```
@Override
public void updateAccountValue(String key, String value, String currency,
        String accountName) {
    System.out.println("UpdateAccountValue. Key: " + key + ", Value: " + value + ", Currency: " +
currency + ", AccountName: " + accountName);
}
```

...

```
@Override
public void updatePortfolio(Contract contract, double position,
        double marketPrice, double marketValue, double averageCost,
        double unrealizedPNL, double realizedPNL, String accountName) {
    System.out.println("UpdatePortfolio. "+contract.symbol()+", "+contract.secType()+" @ "+contract.
exchange()
          +": Position: "+position+", MarketPrice: "+marketPrice+", MarketValue: "+marketValue+",
 AverageCost: "+averageCost
          +", UnrealizedPNL: "+unrealizedPNL+", RealizedPNL: "+realizedPNL+", AccountName: "+
accountName);
}
```

...

```
    @Override
    public void updateAccountTime(String timeStamp) {
        System.out.println("UpdateAccountTime. Time: " + timeStamp+"\n");
    }
```

...

```
    @Override
    public void accountDownloadEnd(String accountName) {
        System.out.println("Account download finished: "+accountName+"\n");
    }
```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

...

```
        Public Sub updateAccountValue(key As String, value As String, currency As String, accountName As
    String) Implements IBApi.EWrapper.updateAccountValue
            Console.WriteLine("UpdateAccountValue. Key: " & key & ", Value: " & value & ", Currency: " &
    currency & ", AccountName: " & accountName)
        End Sub
```

...

```
        Public Sub updatePortfolio(contract As IBApi.Contract, position As Double, marketPrice As Double,
    marketValue As Double, averageCost As Double, unrealizedPNL As Double, realizedPNL As Double, accountName As
    String) Implements IBApi.EWrapper.updatePortfolio
            Console.WriteLine("UpdatePortfolio. " & contract.Symbol & ", " & contract.SecType & " @ " &
    contract.Exchange &
                    ": Position: " & position & ", MarketPrice: " & marketPrice & ", MarketValue: " &
    marketValue & ", AverageCost: " & averageCost &
                    ", UnrealizedPNL: " & unrealizedPNL & ", RealizedPNL: " & realizedPNL & ", AccountName: " &
    accountName)
        End Sub
```

...

```
        Public Sub updateAccountTime(timestamp As String) Implements IBApi.EWrapper.updateAccountTime
            Console.WriteLine("UpdateAccountTime. Time: " & timestamp)
        End Sub
```

...

```
        Public Sub accountDownloadEnd(account As String) Implements IBApi.EWrapper.accountDownloadEnd
            Console.WriteLine("accountDownloadEnd - Account[" & account & "]")
        End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

...

```
void TestCppClient::updateAccountValue(const std::string& key, const std::string& val,
                                       const std::string& currency, const std::string& accountName) {
    printf("UpdateAccountValue. Key: %s, Value: %s, Currency: %s, Account Name: %s\n", key.c_str(), val.
      c_str(), currency.c_str(), accountName.c_str());
}
```

...

```
void TestCppClient::updatePortfolio(const Contract& contract, double position,
                                    double marketPrice, double marketValue, double averageCost,
                                    double unrealizedPNL, double realizedPNL, const std::string&
      accountName){
    printf("UpdatePortfolio. %s, %s @ %s: Position: %g, MarketPrice: %g, MarketValue: %g, AverageCost: %g,
       UnrealizedPNL: %g, RealizedPNL: %g, AccountName: %s\n", (contract.symbol).c_str(), (contract.secType).c_str(
      ), (contract.primaryExchange).c_str(), position, marketPrice, marketValue, averageCost, unrealizedPNL,
      realizedPNL, accountName.c_str());
}
```

...

```
void TestCppClient::updateAccountTime(const std::string& timeStamp) {
    printf( "UpdateAccountTime. Time: %s\n", timeStamp.c_str());
}

...

void TestCppClient::accountDownloadEnd(const std::string& accountName) {
    printf( "Account download finished: %s\n", accountName.c_str());
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

```
...

    def updateAccountValue(self, key: str, val: str, currency: str,
                           accountName: str):
        super().updateAccountValue(key, val, currency, accountName)
        print("UpdateAccountValue. Key:", key, "Value:", val,
              "Currency:", currency, "AccountName:", accountName)


...

    def updatePortfolio(self, contract: Contract, position: float,
                        marketPrice: float, marketValue: float,
                        averageCost: float, unrealizedPNL: float,
                        realizedPNL: float, accountName: str):
        super().updatePortfolio(contract, position, marketPrice, marketValue,
                                averageCost, unrealizedPNL, realizedPNL, accountName)
        print("UpdatePortfolio.", contract.symbol, "", contract.secType, "@",
              contract.exchange, "Position:", position, "MarketPrice:", marketPrice,
              "MarketValue:", marketValue, "AverageCost:", averageCost,
              "UnrealizedPNL:", unrealizedPNL, "RealizedPNL:", realizedPNL,
              "AccountName:", accountName)


...

    def updateAccountTime(self, timeStamp: str):
        super().updateAccountTime(timeStamp)
        print("UpdateAccountTime. Time:", timeStamp)


...

    def accountDownloadEnd(self, accountName: str):
        super().accountDownloadEnd(accountName)
        print("Account download finished:", accountName)
```

13.3.3 Cancelling

Once the subscription to account updates is no longer needed, it can be cancelled by invoking the **IBApi.EClient.↩ reqAccountUpdates** method while specifying the susbcription flag to be False:

- ```
  client.reqAccountUpdates(false, "U150462");
  ```
- ```
  client.reqAccountUpdates(false, "U150462");
  ```
- ```
  client.reqAccountUpdates(False, "U150462")
  ```
- ```
  m_pClient->reqAccountUpdates(false, "U150462");
  ```
- ```
  self.reqAccountUpdates(False, self.account)
  ```

Note: An important key passed back in **IBApi.EWrapper.updateAccountValue** after a call to **IBApi.E↩ Client.reqAccountUpdates** is a boolean value 'accountReady'. If an accountReady value of false is returned that means that the IB server is in the process of resetting at that moment, i.e. the account is 'not ready'. When this occurs subsequent key values returned to **IBApi::EWrapper::updateAccountValue** in the current update can be out of date or incorrect.

Important: only one account at a time can be subscribed at a time. Attempting a second subscription without previously cancelling an active one will not yield any error message although it will override the already subscribed account with the new one. With Financial Advisory (FA) account structures there is an alternative way of specifying the account code such that information is returned for 'All' sub accounts- this is done by appending the letter 'A' to the end of the account number, i.e. reqAccountUpdates(true, "F123456A")

13.3.4 Identifying the Account Keys

Account values delivered via **IBApi.EWrapper.updateAccountValue** can be classified in the following way:

- Commodities: suffixed by a "-C"
- Securities: suffixed by a "-S"
- Totals: no suffix

For further information, please refer to `the Account Window.`

13.3.5 Account Value Update Subscriptions by Model

The **IBApi.EClient.reqAccountUpdatesMulti** can be used with multiple account structures to subscribe to updates in real time for multiple accounts and/or models. As with **IBApi.EClient.reqAccountUpdates** this information is the exact same as the one displayed within the TWS' Account Window. Note this function receives a specific account along with a flag indicating whether to start or stop the subscription.

- ```
  client.reqAccountUpdatesMulti(9002, "U150462", "EUstocks", true);
  ```
- ```
  client.reqAccountUpdatesMulti(9002, "U150462", "EUstocks", true);
  ```
- ```
  client.reqAccountUpdatesMulti(9002, "U150462", "EUstocks", True)
  ```
- ```
  m_pClient->reqAccountUpdatesMulti(9002, "U150462", "EUstocks", true);
  ```
- ```
  self.reqAccountUpdatesMulti(9005, self.account, "", True)
  ```

The resulting account and portfolio information will be delivered via the **IBApi.EWrapper.accountUpdateMulti** and **IBApi.EWrapper.accountUpdateMultiEnd**

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
  public virtual void accountUpdateMulti(int reqId, string account, string modelCode, string key,
  string value, string currency)
  {
  Console.WriteLine("Account Update Multi. Request: " + reqId + ", Account: " + account + ",
  ModelCode: " + modelCode + ", Key: " + key + ", Value: " + value + ", Currency: " + currency + "\n");
  }
  ```

  ...

  ```
  public virtual void accountUpdateMultiEnd(int reqId)
  {
  Console.WriteLine("Account Update Multi End. Request: " + reqId + "\n");
  }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
  @Override
  public void accountUpdateMulti(int reqId, String account, String modelCode,
  String key, String value, String currency) {
  System.out.println("Account Update Multi. Request: " + reqId + ", Account: " + account + ",
  ModelCode: " + modelCode + ", Key: " + key + ", Value: " + value + ", Currency: " + currency + "\n");
  }
  ```

  ...
  ```

```
    @Override
    public void accountUpdateMultiEnd(int reqId) {
        System.out.println("Account Update Multi End. Request: " + reqId + "\n");
    }
```

- ```
    Public Class EWrapperImpl
        Implements EWrapper
  ```

...

```
        Public Sub accountUpdateMulti(requestId As Integer, account As String, modelCode As String, key As
       String, value As String, currency As String) Implements IBApi.EWrapper.accountUpdateMulti
            Console.WriteLine("accountUpdateMulti. Id: " & requestId & ", Account: " & account & ",
       modelCode: " & modelCode & ", key: " & key & ", value: " & value & ", currency: " & currency)
        End Sub
```

...

```
        Public Sub accountUpdateMultiEnd(requestId As Integer) Implements
       IBApi.EWrapper.accountUpdateMultiEnd
            Console.WriteLine("accountUpdateMultiEnd. id: " & requestId)
        End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

...

```
void TestCppClient::accountUpdateMulti( int reqId, const std::string& account, const std::string& modelCode
      , const std::string& key, const std::string& value, const std::string& currency) {
    printf("AccountUpdate Multi. Request: %d, Account: %s, ModelCode: %s, Key, %s, Value: %s, Currency: %s
      \n", reqId, account.c_str(), modelCode.c_str(), key.c_str(), value.c_str(), currency.c_str());
}
```

...

```
void TestCppClient::accountUpdateMultiEnd( int reqId) {
    printf("Account Update Multi End. Request: %d\n", reqId);
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

...

```
    def accountUpdateMulti(self, reqId: int, account: str, modelCode: str,
                           key: str, value: str, currency: str):
        super().accountUpdateMulti(reqId, account, modelCode, key, value,
                                   currency)
        print("Account Update Multi. Request:", reqId, "Account:", account,
              "ModelCode:", modelCode, "Key:", key, "Value:", value,
              "Currency:", currency)
```

...

```
    def accountUpdateMultiEnd(self, reqId: int):
        super().accountUpdateMultiEnd(reqId)
        print("Account Update Multi End. Request:", reqId)
```
```

13.4 Account Summary

13.4.1 Requesting

The **IBApi.EClient.reqAccountSummary** method creates a subscription for the account data displayed in the TWS Account Summary window. It is commonly used with multiple-account structures.

Unlike **IBApi.EClient.reqAccountUpdates** (p. **??**), **IBApi.EClient.reqAccountSummary** can not only re-trieve summarized information for either one or all the managed accounts but also extract only the specified values to be monitored by the client application. The initial invocation of reqAccountSummary will result in a list of all requested values being returned, and then every three minutes those values which have changed will be returned. The update frequency of 3 minutes is the same as the TWS Account Window and cannot be changed.

- `client.reqAccountSummary(9001, "All", AccountSummaryTags.GetAllTags());`
- `client.reqAccountSummary(9001, "All", "AccountType,NetLiquidation,TotalCashValue,SettledCash,AccruedCash,BuyingPower,EquityWithLoanValue,PreviousEquityWithLoanValue,GrossPositionValue,ReqTEquity,ReqTMargin,SMA,InitMarginReq,MaintMarginReq,AvailableFunds,ExcessLiquidity,Cushion,FullInitMarginReq,FullMaintMarginReq,FullAva ,LookAheadMaintMarginReq,LookAheadAvailableFunds,LookAheadExcessLiquidity,HighestSeverity,DayTradesRemaining,Leve`
- `client.reqAccountSummary(9001, "All", AccountSummaryTags.GetAllTags())`
- `m_pClient->reqAccountSummary(9001, "All", AccountSummaryTags::getAllTags());`
- `self.reqAccountSummary(9001, "All", AccountSummaryTags.AllTags)`

Starting from TWS Build 956 and IB Gateway 956, we have added the function to request account summary data (including *CashBalance* and *TotalCashBalance*) for every currency separately using **LEDGER** tags. Please see `TWS Beta Release Notes`.

When the "$LEDGER" tag is specified, the account summary data will be returned in BASE CURRENCY only.

- `client.reqAccountSummary(9002, "All", "$LEDGER");`
- `client.reqAccountSummary(9002, "All", "$LEDGER");`
- `client.reqAccountSummary(9002, "All", "$LEDGER")`
- `m_pClient->reqAccountSummary(9002, "All", "$LEDGER");`
- `self.reqAccountSummary(9002, "All", "$LEDGER")`

When the "$LEDGER:CURRENCY" tag is specified, the account summary data will be returned only in the CUR↩ RENCY specified. The CashBalance and TotalCashBalance returned are the balance in that specific currency only as you see within the TWS Account Window.

Example: "$LEDGER:USD", "$LEDGER:EUR", "$LEDGER:HKD" etc.

- `client.reqAccountSummary(9003, "All", "$LEDGER:EUR");`
- `client.reqAccountSummary(9003, "All", "$LEDGER:EUR");`
- `client.reqAccountSummary(9003, "All", "$LEDGER:EUR")`
- `m_pClient->reqAccountSummary(9003, "All", "$LEDGER:EUR");`
- `self.reqAccountSummary(9003, "All", "$LEDGER:EUR")`

When the "$LEDGER:ALL" tag is specified, the account summary data returned will be summed up values for ALL accounts and currencies.

Example:

Account = All, Currency = EUR, CashBalance = 12345.67

Account = All, Currency = JPY, CashBalance = 987.54

- `client.reqAccountSummary(9004, "All", "$LEDGER:ALL");`
- `client.reqAccountSummary(9004, "All", "$LEDGER:ALL");`
- `client.reqAccountSummary(9004, "All", "$LEDGER:ALL")`
- `m_pClient->reqAccountSummary(9004, "All", "$LEDGER:ALL");`
- `self.reqAccountSummary(9004, "All", "$LEDGER:ALL")`

Important: only **two** active summary subscriptions are allowed at a time!

13.4.2 Receiving

Summarised information is delivered via **IBApi.EWrapper.accountSummary** and **IBApi.EWrapper.↩ accountSummaryEnd**

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
      public virtual void accountSummary(int reqId, string account, string tag, string value, string
    currency)
      {
          Console.WriteLine("Acct Summary. ReqId: " + reqId + ", Acct: " + account + ", Tag: " + tag + ",
     Value: " + value + ", Currency: " + currency);
      }
  ```

  ...

  ```
      public virtual void accountSummaryEnd(int reqId)
      {
          Console.WriteLine("AccountSummaryEnd. Req Id: "+reqId+"\n");
      }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
  @Override
  public void accountSummary(int reqId, String account, String tag,
          String value, String currency) {
      System.out.println("Acct Summary. ReqId: " + reqId + ", Acct: " + account + ", Tag: " + tag + ",
    Value: " + value + ", Currency: " + currency);
  }
  ```

  ...

  ```
  @Override
  public void accountSummaryEnd(int reqId) {
      System.out.println("AccountSummaryEnd. Req Id: "+reqId+"\n");
  }
  ```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

  ...

  ```
      Public Sub accountSummary(reqId As Integer, account As String, tag As String, value As String,
    currency As String) Implements IBApi.EWrapper.accountSummary
          Console.WriteLine("AccountSummary - ReqId [" & reqId & "] Account [" & account & "] Tag [" &
    tag & "] Value [" & value &
                      "] Currency [" & currency & "]")
      End Sub
  ```

  ...

  ```
      Public Sub accountSummaryEnd(reqId As Integer) Implements IBApi.EWrapper.accountSummaryEnd
          Console.WriteLine("AccountSummaryEnd - ReqId [" & reqId & "]")
      End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::accountSummary( int reqId, const std::string& account, const std::string& tag, const
      std::string& value, const std::string& currency) {
    printf( "Acct Summary. ReqId: %d, Account: %s, Tag: %s, Value: %s, Currency: %s\n", reqId, account.
      c_str(), tag.c_str(), value.c_str(), currency.c_str());
  }
  ```

  ...

```
void TestCppClient::accountSummaryEnd( int reqId) {
    printf( "AccountSummaryEnd. Req Id: %d\n", reqId);
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def accountSummary(self, reqId: int, account: str, tag: str, value: str,
                         currency: str):
          super().accountSummary(reqId, account, tag, value, currency)
          print("Acct Summary. ReqId:", reqId, "Acct:", account,
                "Tag: ", tag, "Value:", value, "Currency:", currency)
  ```

  ...

  ```
      def accountSummaryEnd(self, reqId: int):
          super().accountSummaryEnd(reqId)
          print("AccountSummaryEnd. Req Id: ", reqId)
  ```

13.4.3 Cancelling

Once the subscription to account summary is no longer needed, it can be cancelled via the **IBApi::EClient::cancelAccountSummary** method:

- ```
              client.cancelAccountSummary(9001);
              client.cancelAccountSummary(9002);
              client.cancelAccountSummary(9003);
              client.cancelAccountSummary(9004);
  ```

- ```
          client.cancelAccountSummary(9001);
          client.cancelAccountSummary(9002);
          client.cancelAccountSummary(9003);
          client.cancelAccountSummary(9004);
  ```

- ```
          client.cancelAccountSummary(9001)
          client.cancelAccountSummary(9002)
          client.cancelAccountSummary(9003)
          client.cancelAccountSummary(9004)
  ```

- ```
      m_pClient->cancelAccountSummary(9001);
      m_pClient->cancelAccountSummary(9002);
      m_pClient->cancelAccountSummary(9003);
      m_pClient->cancelAccountSummary(9004);
  ```

- ```
          self.cancelAccountSummary(9001)
          self.cancelAccountSummary(9002)
          self.cancelAccountSummary(9003)
          self.cancelAccountSummary(9004)
  ```

13.5 Positions

13.5.1 Requesting

A limitation of the function **IBApi.EClient.reqAccountUpdates** is that it can only be used with a single account at a time. To create a subscription for position updates from multiple accounts, the function **IBApi.EClient.reqPositions** is available. After initially invoking reqPositions, information about all positions in all associated accounts will be returned, followed by the **IBApi::EWrapper::positionEnd** callback. Thereafter, when a position has changed an update will be returned to the **IBApi::EWrapper::position** function. To cancel a reqPositions subscription, invoke **IBApi::EClient::cancelPositions** (p. **??**).

- ```
            client.reqPositions();
  ```
- ```
        client.reqPositions();
  ```
- ```
        client.reqPositions()
  ```
- ```
    m_pClient->reqPositions();
  ```
- ```
        self.reqPositions()
  ```

13.5.2 Receiving

After invoking the above, the positions will then be received through the **IBApi.EWrapper.position** callback. After the **initial callback** (only) of all positions, the **IBApi.EWrapper.positionEnd** function will be triggered.

- For futures, the exchange field will not be populated in the position callback as some futures trade on multiple exchanges

- ```
    public class EWrapperImpl : EWrapper
    {
  ```

  ...

  ```
        public virtual void position(string account, Contract contract, double pos, double avgCost)
        {
            Console.WriteLine("Position. "+account+" - Symbol: "+contract.Symbol+", SecType: "+contract.
      SecType+", Currency: "+contract.Currency+", Position: "+pos+", Avg cost: "+avgCost);
        }
  ```

  ...

  ```
        public virtual void positionEnd()
        {
            Console.WriteLine("PositionEnd \n");
        }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
    @Override
    public void position(String account, Contract contract, double pos,
            double avgCost) {
        System.out.println("Position. "+account+" - Symbol: "+contract.symbol()+", SecType: "+contract.
      secType()+", Currency: "+contract.currency()+", Position: "+pos+", Avg cost: "+avgCost);
    }
  ```

  ...

  ```
    @Override
    public void positionEnd() {
        System.out.println("PositionEnd \n");
    }
  ```

- ```
    Public Class EWrapperImpl
        Implements EWrapper
  ```

  ...

  ```
        Public Sub position(account As String, contract As IBApi.Contract, pos As Double, avgCost As
      Double) Implements IBApi.EWrapper.position
            Console.WriteLine("Position. " & account & " - Symbol: " & contract.Symbol & ", SecType: " &
      contract.SecType & ", Currency: " &
                              contract.Currency & ", Position: " & pos & ", Avg cost: " & avgCost)
        End Sub
  ```

  ...

```
        Public Sub positionEnd() Implements IBApi.EWrapper.positionEnd
            Console.WriteLine("PositionEnd \n")
        End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::position( const std::string& account, const Contract& contract, double position, double
         avgCost) {
      printf( "Position. %s - Symbol: %s, SecType: %s, Currency: %s, Position: %g, Avg Cost: %g\n", account.
        c_str(), contract.symbol.c_str(), contract.secType.c_str(), contract.currency.c_str(), position, avgCost);
  }
  ```

  ...

  ```
  void TestCppClient::positionEnd() {
      printf( "PositionEnd\n");
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def position(self, account: str, contract: Contract, position: float,
                   avgCost: float):
          super().position(account, contract, position, avgCost)
          print("Position.", account, "Symbol:", contract.symbol, "SecType:",
                contract.secType, "Currency:", contract.currency,
                "Position:", position, "Avg cost:", avgCost)
  ```

  ...

  ```
      def positionEnd(self):
          super().positionEnd()
          print("PositionEnd")
  ```

13.5.3 Cancelling

To cancel the reqPosition subscription, invoke **IBApi::EClient::cancelPositions** (p. **??**):

- ```
              client.cancelPositions();
  ```
- ```
          client.cancelPositions();
  ```
- ```
          client.cancelPositions()
  ```
- ```
      m_pClient->cancelPositions();
  ```
- ```
          self.cancelPositions()
  ```

13.5.4 Position Update Subscription by Model

The function **IBApi.EClient.reqPositionsMulti** can be used with multiple account structures to subscribe to positions updates for multiple accounts and/or models. The account and model parameters are optional.

- ```
              client.reqPositionsMulti(9003, "DU74649", "EUstocks");
  ```
- ```
          client.reqPositionsMulti(9003, "DU74649", "EUstocks");
  ```
- ```
          client.reqPositionsMulti(9003, "DU74649", "EUstocks")
  ```
- ```
      m_pClient->reqPositionsMulti(9003, "U150462", "EUstocks");
  ```
- ```
          self.reqPositionsMulti(9006, self.account, "")
  ```

After invoking **IBApi.EClient.reqPositionsMulti** data will be returned to the **IBApi.EWrapper.positionMulti** function. After the initial callback of all positions matching the supplied criteria to reqPositionsMulti, the **IB↩ Api.EWrapper.positionMultiEnd** function will be triggered. Thereafter, there will only be messages sent to positionsMulti when there is a change.

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
      public virtual void positionMulti(int reqId, string account, string modelCode, Contract contract,
    double pos, double avgCost)
      {
          Console.WriteLine("Position Multi. Request: " + reqId + ", Account: " + account + ", ModelCode:
   " + modelCode + ", Symbol: " + contract.Symbol + ", SecType: " + contract.SecType + ", Currency: " +
    contract.Currency + ", Position: " + pos + ", Avg cost: " + avgCost + "\n");
      }
  ```

  ...

  ```
      public virtual void positionMultiEnd(int reqId)
      {
          Console.WriteLine("Position Multi End. Request: " + reqId + "\n");
      }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
  @Override
  public void positionMulti(int reqId, String account, String modelCode,
          Contract contract, double pos, double avgCost) {
      System.out.println("Position Multi. Request: " + reqId + ", Account: " + account + ", ModelCode: "
    + modelCode + ", Symbol: " + contract.symbol() + ", SecType: " + contract.secType() + ", Currency: " +
    contract.currency() + ", Position: " + pos + ", Avg cost: " + avgCost + "\n");
  }
  ```

  ...

  ```
  @Override
  public void positionMultiEnd(int reqId) {
      System.out.println("Position Multi End. Request: " + reqId + "\n");
  }
  ```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

  ...

  ```
      Public Sub positionMulti(requestId As Integer, account As String, modelCode As String, contract As
    Contract, pos As Double, avgCost As Double) Implements IBApi.EWrapper.positionMulti
          Console.WriteLine("positionMulti. Id: " & requestId & ", Account: " & account & ", ModelCode: "
    & modelCode & ", Contract: " & contract.Symbol & ", pos: " & pos & ", avgCost: " & avgCost)
      End Sub
  ```

  ...

  ```
      Public Sub positionMultiEnd(requestId As Integer) Implements IBApi.EWrapper.positionMultiEnd
          Console.WriteLine("positionMultiEnd \n")
      End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::positionMulti( int reqId, const std::string& account,const std::string& modelCode,
        const Contract& contract, double pos, double avgCost) {
      printf("Position Multi. Request: %d, Account: %s, ModelCode: %s, Symbol: %s, SecType: %s, Currency: %s,
       Position: %g, Avg Cost: %g\n", reqId, account.c_str(), modelCode.c_str(), contract.symbol.c_str(), contract
      .secType.c_str(), contract.currency.c_str(), pos, avgCost);
  }
  ```

...

```
void TestCppClient::positionMultiEnd( int reqId) {
    printf("Position Multi End. Request: %d\n", reqId);
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

...

```
    def positionMulti(self, reqId: int, account: str, modelCode: str,
                      contract: Contract, pos: float, avgCost: float):
        super().positionMulti(reqId, account, modelCode, contract, pos, avgCost)
        print("Position Multi. Request:", reqId, "Account:", account,
              "ModelCode:", modelCode, "Symbol:", contract.symbol, "SecType:",
              contract.secType, "Currency:", contract.currency, ",Position:",
              pos, "AvgCost:", avgCost)


...

    def positionMultiEnd(self, reqId: int):
        super().positionMultiEnd(reqId)
        print("Position Multi End. Request:", reqId)
```

13.6 Profit And Loss (P&L)

13.6.1 P&L data in the Account Window

UnRealized and Realized P&L is sent to the API function **IBApi.EWrapper.updateAccountValue** function after a subscription request is made with **IBApi.EClient.reqAccountUpdates** (p. **??**). This information corresponds to the data in the TWS Account Window, and has a different source of information, a different update frequency, and different reset schedule than PnL data in the TWS Portfolio Window and associated API functions (below). In particular, the unrealized P&L information shown in the TWS Account Window which is sent to updatePortfolioValue will update either **(1)** when a trade for that particular instrument occurs or **(2)** every 3 mintues. The realized P&L data in the TWS `Account Window` is reset to 0 once per day.

- It is important to keep in mind that the P&L data shown in the Account Window and Portfolio Window will sometimes differ because there is a different source of information and a different reset schedule.

13.6.2 P&L data in the Portfolio Window

Beginning with API v973.03, requests can be made to receive real time updates about the daily P&L and unrealized P&L for an account, or for individual positions. Financial Advisors can also request P&L figures for a FA group, or portfolio model. In API v973.05+/TWS v968+ this is further extended to include realized P&L information at the account or individual position level.

These newer P&L API functions demonstrated below return the data which is displayed in the TWS `Portfolio Window` in current versions of TWS (v963+). As such, the P&L values are calculated based on the reset schedule specifed in TWS Global Configuration (by default an instrument-specific reset schedule) and this setting affects values sent to the associated API functions as well. Also in TWS, P&L data from virtual forex positions will be included in the account P&L if and only if the Virtual Fx section of the Account Window is expanded.

-
-
-
-
- **Note:** the P&L functions in Python API are available starting in API v973.06+.
```

P&L subscription requests for individual positions

Subscribe using the **IBApi::EClient::reqPnLSingle** function

- ```
            client.reqPnLSingle(17001, "DUD00029", "", 268084);
  ```
- ```
        client.reqPnLSingle(17001, "DUD00029", "", 268084);
  ```
- ```
        client.reqPnLSingle(17001, "DUD00029", "", 268084)
  ```
- ```
    m_pClient->reqPnLSingle(7002, "DUD00029", "", 268084);
  ```
- ```
        self.reqPnLSingle(17002, "DU242650", "", 265598);
  ```

Currently updates are returned to **IBApi.EWrapper.pnlSingle** approximately once per second. ∗subject to change in the future

- ```
        public void pnlSingle(int reqId, int pos, double dailyPnL, double unrealizedPnL, double realizedPnL
      , double value)
        {
            Console.WriteLine("PnL Single. Request Id: {0}, Pos {1}, Daily PnL {2}, Unrealized PnL {3},
     Realized PnL: {4}, Value: {5}", reqId, pos, dailyPnL, unrealizedPnL, realizedPnL, value);
        }
  ```
- ```
    @Override
    public void pnlSingle(int reqId, int pos, double dailyPnL, double unrealizedPnL, double realizedPnL,
      double value) {
        System.out.println(EWrapperMsgGenerator.pnlSingle(reqId, pos, dailyPnL, unrealizedPnL, realizedPnL,
     value));
    }
  ```
- ```
        Public Sub pnlSingle(reqId As Integer, pos As Integer, dailyPnL As Double, unrealizedPnL As Double,
     realizedPnL As Double, value As Double) Implements EWrapper.pnlSingle
            Console.WriteLine("PnL Single. Request Id: {0}, pos: {1}, daily PnL: {2}, unrealized PnL: {3},
     realized PnL: {4}, value: {5}", reqId, pos, dailyPnL, unrealizedPnL, realizedPnL, value)
        End Sub
  ```
- ```
void TestCppClient::pnlSingle(int reqId, int pos, double dailyPnL, double unrealizedPnL, double realizedPnL
      , double value) {
    printf("PnL Single. ReqId: %d, pos: %d, daily PnL: %g, unrealized PnL: %g, realized PnL: %g, value: %g
      \n", reqId, pos, dailyPnL, unrealizedPnL, realizedPnL, value);
}
  ```
- ```
    def pnlSingle(self, reqId: int, pos: int, dailyPnL: float,
                  unrealizedPnL: float, realizedPnL: float, value: float):
        super().pnlSingle(reqId, pos, dailyPnL, unrealizedPnL, realizedPnL, value)
        print("Daily PnL Single. Req Id: ", reqId, ", pos: ", pos,
              ", daily PnL: ", dailyPnL, ", unrealizedPnL: ", unrealizedPnL,
              ", realizedPnL: ", realizedPnL, ", value: ", value)
  ```

- If a P&L subscription request is made for an invalid conId or contract not in the account, there will not be a response.
- As elsewhere in the API, a max double value will indicate an 'unset' value. This corresponds to an empty cell in TWS.

Subscriptions are cancelled using the **IBApi::EClient::cancelPnLSingle** function:

- ```
            client.cancelPnLSingle(17001);
  ```
- ```
        client.cancelPnLSingle(17001);
  ```
- ```
        client.cancelPnLSingle(17001)
  ```
- ```
    m_pClient->cancelPnLSingle(7002);
  ```
- ```
        self.cancelPnLSingle(17002);
  ```
  ```

P&L subscription requests for accounts

Subscribe using the **IBApi::EClient::reqPnL** function:

```
            client.reqPnL(17001, "DUD00029", "");
```

```
        client.reqPnL(17001, "DUD00029", "");
```

```
        client.reqPnL(17001, "DUD00029", "")
```

```
    m_pClient->reqPnL(7001, "DUD00029", "");
```

```
        self.reqPnL(17001, "DU242650", "")
```

- For FA accounts, if 'All' is specified as the accound code, aggregated data for all subaccounts is returned.
- With requests for advisor accounts with many subaccounts and/or positions can take several seconds for aggregated P&L to be computed and returned.
- For account P&L data the TWS setting "Prepare portfolio PnL data when downloading positions" must be checked.

Updates are sent to **IBApi.EWrapper.pnl**

```
        public void pnl(int reqId, double dailyPnL, double unrealizedPnL, double realizedPnL)
        {
            Console.WriteLine("PnL. Request Id: {0}, Daily PnL: {1}, Unrealized PnL: {2}, Realized PnL: {3}
      ", reqId, dailyPnL, unrealizedPnL, realizedPnL);
        }
```

```
    @Override
    public void pnl(int reqId, double dailyPnL, double unrealizedPnL, double realizedPnL) {
        System.out.println(EWrapperMsgGenerator.pnl(reqId, dailyPnL, unrealizedPnL, realizedPnL));
    }
```

```
        Public Sub pnl(reqId As Integer, dailyPnL As Double, unrealizedPnL As Double, realizedPnL As
     Double) Implements EWrapper.pnl
            Console.WriteLine("PnL. Request Id: {0}, daily PnL: {1}, unrealized PnL: {2}, realized PnL:
     {3}", reqId, dailyPnL, unrealizedPnL, realizedPnL)
        End Sub
```

```
void TestCppClient::pnl(int reqId, double dailyPnL, double unrealizedPnL, double realizedPnL) {
    printf("PnL. ReqId: %d, daily PnL: %g, unrealized PnL: %g, realized PnL: %g\n", reqId, dailyPnL,
      unrealizedPnL, realizedPnL);
}
```

```
    def pnl(self, reqId: int, dailyPnL: float,
            unrealizedPnL: float, realizedPnL: float):
        super().pnl(reqId, dailyPnL, unrealizedPnL, realizedPnL)
        print("Daily PnL. Req Id: ", reqId, ", daily PnL: ", dailyPnL,
              ", unrealizedPnL: ", unrealizedPnL, ", realizedPnL: ", realizedPnL)
```

Cancel unnecessary subscriptions with the **IBApi::EClient::cancelPnL** function:

```
            client.cancelPnL(17001);
```

```
        client.cancelPnL(17001);
```

```
        client.cancelPnL(17001)
```

```
    m_pClient->cancelPnL(7001);
```

```
        self.cancelPnL(17001)
```

Chapter 13

Options

http://interactivebrokers.github.io/tws-api/options.html

14.1 Option Chains

The option chain for a given security can be returned using the function reqContractDetails. If an option contract is incompletely defined (for instance with the strike undefined) and used as an argument to **IBApi::EClient::req↩ ContractDetails** (p. **??**), a list of all matching option contracts will be returned.

The example below shows an "incomplete" option **IBApi.Contract** with no last trading day, strike nor multi-plier defined. In most cases using such a contract would result into a contract ambiguity error since there are lots of instruments matching the same description. **IBApi.EClient.reqContractDetails** will instead use it to obtain the whole option chain from the TWS.

-
```
            Contract contract = new Contract();
            contract.Symbol = "FISV";
            contract.SecType = "OPT";
            contract.Exchange = "SMART";
            contract.Currency = "USD";
```

  ...

```
            client.reqContractDetails(209, ContractSamples.OptionForQuery());
            client.reqContractDetails(210, ContractSamples.EurGbpFx());
            client.reqContractDetails(211, ContractSamples.Bond());
            client.reqContractDetails(212, ContractSamples.FuturesOnOptions());
            client.reqContractDetails(213, ContractSamples.SimpleFuture());
```

-
```
        Contract contract = new Contract();
        contract.symbol("FISV");
        contract.secType("OPT");
        contract.currency("USD");
        contract.exchange("SMART");
```

  ...

```
        client.reqContractDetails(210, ContractSamples.OptionForQuery());
        client.reqContractDetails(211, ContractSamples.EurGbpFx());
        client.reqContractDetails(212, ContractSamples.Bond());
        client.reqContractDetails(213, ContractSamples.FuturesOnOptions());
        client.reqContractDetails(214, ContractSamples.SimpleFuture());
```

-
```
            Dim contract As Contract = New Contract
            contract.Symbol = "FISV"
            contract.SecType = "OPT"
            contract.Exchange = "SMART"
            contract.Currency = "USD"
```

  ...

```
            client.reqContractDetails(209, ContractSamples.EurGbpFx())
            client.reqContractDetails(210, ContractSamples.OptionForQuery())
            client.reqContractDetails(211, ContractSamples.Bond())
            client.reqContractDetails(212, ContractSamples.FuturesOnOptions())
            client.reqContractDetails(213, ContractSamples.SimpleFuture())
```

- ```
    Contract contract;
    contract.symbol = "FISV";
    contract.secType = "OPT";
    contract.exchange = "SMART";
    contract.currency = "USD";
  ```

...

```
    m_pClient->reqContractDetails(210, ContractSamples::OptionForQuery());
    m_pClient->reqContractDetails(212, ContractSamples::IBMBond());
    m_pClient->reqContractDetails(213, ContractSamples::IBKRStk());
    m_pClient->reqContractDetails(214, ContractSamples::Bond());
    m_pClient->reqContractDetails(215, ContractSamples::FuturesOnOptions());
    m_pClient->reqContractDetails(216, ContractSamples::SimpleFuture());
```

- ```
        contract = Contract()
        contract.symbol = "FISV"
        contract.secType = "OPT"
        contract.exchange = "SMART"
        contract.currency = "USD"
  ```

...

```
        self.reqContractDetails(209, ContractSamples.EurGbpFx())
        self.reqContractDetails(210, ContractSamples.OptionForQuery())
        self.reqContractDetails(211, ContractSamples.Bond())
        self.reqContractDetails(212, ContractSamples.FuturesOnOptions())
```

One limitation of this technique is that the return of option chains will be throttled and take a longer time the more ambiguous the contract definition. Starting in version 9.72 of the API, a new function **IBApi::EClient::reqSecDef↩ OptParams** is introduced that does not have the throttling limitation.

- It is not recommended to use reqContractDetails to receive complete option chains on an underlying, e.g. all combinations of strikes/rights/expiries.
- For very large option chains returned from reqContractDetails, unchecking the setting in TWS Global Configuration at API -> Settings -> "Expose entire trading schedule to the API" will decrease the amount of data returned per option and help to return the contract list more quickly.

- ```
            client.reqSecDefOptParams(0, "IBM", "", "STK", 8314);
  ```

...

```
        public void securityDefinitionOptionParameter(int reqId, string exchange, int underlyingConId,
      string tradingClass, string multiplier, HashSet<string> expirations, HashSet<double> strikes)
        {
            Console.WriteLine("Security Definition Option Parameter. Reqest: {0}, Exchange: {1}, Undrelying
     contract id: {2}, Trading class: {3}, Multiplier: {4}, Expirations: {5}, Strikes: {6}",
                              reqId, exchange, underlyingConId, tradingClass, multiplier, string.Join(", ",
     expirations), string.Join(", ", strikes));
        }
```

- ```
        client.reqSecDefOptParams(0, "IBM", "", "STK", 8314);
  ```

...

```
    @Override
    public void securityDefinitionOptionalParameter(int reqId, String exchange,
            int underlyingConId, String tradingClass, String multiplier,
            Set<String> expirations, Set<Double> strikes) {
        System.out.println("Security Definition Optional Parameter. Request: "+reqId+", Trading Class: "+
      tradingClass+", Multiplier: "+multiplier+" \n");
    }
```

- ```
        client.reqSecDefOptParams(0, "IBM", "", "STK", 8314)
  ```

...
  ```

```
         Public Sub securityDefinitionOptionParameter(reqId As Integer, exchange As String, underlyingConId
        As Integer, tradingClass As String, multiplier As String, expirations As HashSet(Of String), strikes As
        HashSet(Of Double)) Implements EWrapper.securityDefinitionOptionParameter
             Console.WriteLine("securityDefinitionOptionParameter: " & reqId & " tradingClass: " &
        tradingClass & " multiplier: ")
         End Sub
```

- ```
    m_pClient->reqSecDefOptParams(0, "IBM", "", "STK", 8314);
  ```

  ...

  ```
  void TestCppClient::securityDefinitionOptionalParameter(int reqId, const std::string& exchange, int
        underlyingConId, const std::string& tradingClass,
                                                          const std::string& multiplier, const
        std::set<std::string>& expirations, const std::set<double>& strikes) {
      printf("Security Definition Optional Parameter. Request: %d, Trading Class: %s, Multiplier: %s\n",
        reqId, tradingClass.c_str(), multiplier.c_str());
  }
  ```

- ```
      self.reqSecDefOptParams(0, "IBM", "", "STK", 8314)
  ```

  ...

  ```
      def securityDefinitionOptionParameter(self, reqId: int, exchange: str,
                                            underlyingConId: int, tradingClass: str, multiplier: str,
                                            expirations: SetOfString, strikes: SetOfFloat):
          super().securityDefinitionOptionParameter(reqId, exchange,
                                                    underlyingConId, tradingClass, multiplier, expirations,
        strikes)
          print("Security Definition Option Parameter. ReqId:%d Exchange:%s "
                "Underlying conId: %d TradingClass:%s Multiplier:%s Exp:%s Strikes:%s",
                reqId, exchange, underlyingConId, tradingClass, multiplier,
                ",".join(expirations), ",".join(str(strikes)))
  ```

IBApi::EClient::reqSecDefOptParams returns a list of expiries and a list of strike prices. In some cases it is possible there are combinations of strike and expiry that would not give a valid option contract.

The API can return the greek values in real time for options, as well as calculate the implied volatility given a hypothetical price or calculate the hypothetical price given an implied volatility.

- **Option Greeks**

14.2 Exercising options

Options are exercised or lapsed from the API with the function **IBApi.EClient.exerciseOptions**

- Option exercise will appear with order status side = "BUY" and limit price of 0, but only at the time the request is made
- Option exercise can be distinguished by price = 0

- ```
      client.exerciseOptions(5003, ContractSamples.OptionWithTradingClass(), 1, 1, null, 1);
  ```

- ```
      //** Exercising options ***
      client.exerciseOptions(5003, ContractSamples.OptionWithTradingClass(), 1, 1, "", 1);
  ```

- ```
      '** Exercising options ***
      client.exerciseOptions(5003, ContractSamples.OptionWithTradingClass(), 1, 1, Nothing, 1)
  ```

- ```
  //** Exercising options ***
  m_pClient->exerciseOptions(5003, ContractSamples::OptionWithTradingClass(), 1, 1, "", 1);
  ```

- ```
      self.exerciseOptions(5003, ContractSamples.OptionWithTradingClass(), 1,
                           1, self.account, 1)
  ```
  ```

14.3 Option Greeks

14.3.1 Market data related to options

The option greek values- delta, gamma, theta, vega- are returned by default following a reqMktData() request for the option. See **Available Tick Types**

Tick types "Bid Option Computation" (#10), "Ask Option Computation" (#11), "Last Option Computation" (#12), and "Model Option Computation" (#13) return all Greeks (delta, gamma, vega, theta), the underlying price and the stock and option reference price when requested.

MODEL_OPTION_COMPUTATION also returns model implied volatility.

Note that to receive live greek values it is necessary to have market data subscriptions for **both** the option and the underlying contract.

- ```
            //Requesting data for an option contract will return the greek values
            client.reqMktData(1002, ContractSamples.OptionWithLocalSymbol(), string.Empty, false, false,
      null);
  ```
- ```
        //Requesting data for an option contract will return the greek values
        client.reqMktData(1002, ContractSamples.OptionWithLocalSymbol(), "", false, false, null);
  ```
- ```
        'Requesting data for an option contract will return the greek values
        client.reqMktData(1005, ContractSamples.USOptionContract(), String.Empty, False, False, Nothing)
  ```
- ```
    //Requesting data for an option contract will return the greek values
    m_pClient->reqMktData(1013, ContractSamples::USOptionContract(), "", false, false, TagValueListSPtr());
  ```
- ```
        # Requesting data for an option contract will return the greek values
        self.reqMktData(1013, ContractSamples.OptionWithLocalSymbol(), "", False, False, [])
  ```

The Implied Volatility of an underlying based on its current option prices is returned in tick 24. See **Available Tick Types**

The IB 30-day volatility is the at-market volatility estimated for a maturity thirty calendar days forward of the current trading day, and is based on option prices from two consecutive expiration months.

14.3.2 Calculating option prices and historical volatility

The implied volatility for an option given its price and the price of the underlying can be calculated with the function **IBApi.EClient.calculateImpliedVolatility**

- ```
            client.calculateImpliedVolatility(5001, ContractSamples.OptionAtBOX(), 5, 85, null);
  ```
- ```
        client.calculateImpliedVolatility(5001, ContractSamples.OptionAtBOX(), 5, 85, null);
  ```
- ```
        '** Calculating implied volatility ***
        client.calculateImpliedVolatility(5001, ContractSamples.NormalOption(), 5, 85, Nothing)
        '** Canceling implied volatility ***
        client.cancelCalculateImpliedVolatility(5001)
  ```
- ```
    m_pClient->calculateImpliedVolatility(5001, ContractSamples::NormalOption(), 5, 85, TagValueListSPtr())
      ;
  ```
- ```
        self.calculateImpliedVolatility(5001, ContractSamples.OptionAtBOX(), 5, 85, [])
  ```

Alternatively, given the price of the underlying and an implied volatility it is possible to calculate the option price using the function **IBApi.EClient.calculateOptionPrice**

```
            client.calculateOptionPrice(5002, ContractSamples.OptionAtBOX(), 0.22, 85, null);
```

```
        client.calculateOptionPrice(5002, ContractSamples.OptionAtBOX(), 0.22, 85, null);
```

```
        '** Calculating option's price ***
        client.calculateOptionPrice(5002, ContractSamples.NormalOption(), 0.22, 85, Nothing)
        '** Canceling option's price calculation ***
        client.cancelCalculateOptionPrice(5002)
```

```
    m_pClient->calculateOptionPrice(5002, ContractSamples::NormalOption(), 0.22, 85, TagValueListSPtr());
```

```
        self.calculateOptionPrice(5002, ContractSamples.OptionAtBOX(), 0.22, 85, [])
```

After the request, the option specific information will be delivered via the **IBApi.EWrapper.tickOptionComputation** method:

```
    public class EWrapperImpl : EWrapper
    {
```

...

```
        public virtual void tickOptionComputation(int tickerId, int field, double impliedVolatility, double
       delta, double optPrice, double pvDividend, double gamma, double vega, double theta, double undPrice)
        {
            Console.WriteLine("TickOptionComputation. TickerId: "+tickerId+", field: "+field+",
       ImpliedVolatility: "+impliedVolatility+", Delta: "+delta
                +", OptionPrice: "+optPrice+", pvDividend: "+pvDividend+", Gamma: "+gamma+", Vega: "+vega+"
     , Theta: "+theta+", UnderlyingPrice: "+undPrice);
        }
```

```
public class EWrapperImpl implements EWrapper {
```

...

```
    @Override
    public void tickOptionComputation(int tickerId, int field,
            double impliedVol, double delta, double optPrice,
            double pvDividend, double gamma, double vega, double theta,
            double undPrice) {
        System.out.println("TickOptionComputation. TickerId: "+tickerId+", field: "+field+",
       ImpliedVolatility: "+impliedVol+", Delta: "+delta
                +", OptionPrice: "+optPrice+", pvDividend: "+pvDividend+", Gamma: "+gamma+", Vega: "+vega+"
     , Theta: "+theta+", UnderlyingPrice: "+undPrice);
    }
```

```
    Public Class EWrapperImpl
        Implements EWrapper
```

...

```
        Public Sub tickOptionComputation(tickerId As Integer, field As Integer, impliedVolatility As
       Double, delta As Double, optPrice As Double, pvDividend As Double, gamma As Double, vega As Double, theta As
       Double, undPrice As Double) Implements IBApi.EWrapper.tickOptionComputation
            Console.WriteLine("TickOptionComputation. TickerId: " & tickerId & ", field: " & field & ",
       ImpliedVolatility: " & impliedVolatility &
                              ", Delta: " & delta & ", OptionPrice: " & optPrice & ", pvDividend: " & pvDividend
       & ", Gamma: " & gamma &
                             ", Vega: " & vega & ", Theta: " & theta & ", UnderlyingPrice: " & undPrice)
        End Sub
```

```
class TestCppClient : public EWrapper
{
```

...

```
void TestCppClient::tickOptionComputation( TickerId tickerId, TickType tickType, double impliedVol, double
      delta,
                                          double optPrice, double pvDividend,
                                          double gamma, double vega, double theta, double undPrice) {
    printf( "TickOptionComputation. Ticker Id: %ld, Type: %d, ImpliedVolatility: %g, Delta: %g,
       OptionPrice: %g, pvDividend: %g, Gamma: %g, Vega: %g, Theta: %g, Underlying Price: %g\n", tickerId, (int)tickType
      impliedVol, delta, optPrice, pvDividend, gamma, vega, theta, undPrice);
}
```

```
class TestWrapper(wrapper.EWrapper):
```

...

```
def tickOptionComputation(self, reqId: TickerId, tickType: TickType,
                          impliedVol: float, delta: float, optPrice: float, pvDividend: float,
                          gamma: float, vega: float, theta: float, undPrice: float):
    super().tickOptionComputation(reqId, tickType, impliedVol, delta,
                                  optPrice, pvDividend, gamma, vega, theta, undPrice)
    print("TickOptionComputation. TickerId:", reqId, "tickType:", tickType,
          "ImpliedVolatility:", impliedVol, "Delta:", delta, "OptionPrice:",
          optPrice, "pvDividend:", pvDividend, "Gamma: ", gamma, "Vega:", vega,
          "Theta:", theta, "UnderlyingPrice:", undPrice)
```

Chapter 14

Financial Advisors

http://interactivebrokers.github.io/tws-api/financial_advisor.html

15.1 Allocation Methods and Groups

A number of methods and profiles are available with Financial Advisor and IBroker account structures to specify how trades should be distributed across multiple accounts. This functionality allows for trades to be placed across multiple accounts. The API has the same functionality available as TWS.

Group and Profile order allocation methods for Financial Advisor Accounts can be created directly in TWS↩ : `Allocations and Transfers`, or utilize the **IBApi.EClient.replaceFA()** method via the API directly.

As suggested from the method names below, a Group will distribute the order based on inherent properties such as the account's liquidation value or equity whereas a Profile will offer the possibility to assign the allocated proportion based on explicit ratios or percentages.

Allocation by Group

EqualQuantity

Requires you to specify an order size. This method distributes shares equally between all accounts in the group.

Example: You transmit an order for 400 shares of stock ABC. If your Account Group includes four accounts, each account receives 100 shares. If your Account Group includes six accounts, each account receives 66 shares, and then 1 share is allocated to each account until all are distributed.

NetLiq

Requires you to specify an order size. This method distributes shares based on the net liquidation value of each account. The system calculates ratios based on the Net Liquidation value in each account and allocates shares based on these ratios.

Example: You transmit an order for 700 shares of stock XYZ. The account group includes three accounts, A, B and C with Net Liquidation values of $25,000, $50,000 and $100,000 respectively. The system calculates a ratio of 1:2:4 and allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

AvailableEquity

Requires you to specify an order size. This method distributes shares based on the amount of available equity in each account. The system calculates ratios based on the Available Equity in each account and allocates shares based on these ratios.

Example: You transmit an order for 700 shares of stock XYZ. The account group includes three accounts, A, B and C with available equity in the amounts of $25,000, $50,000 and $100,000 respectively. The system calculates a ratio of 1:2:4 and allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

PctChange

Do not specify an order size. Since the quantity is calculated by the system, the order size is displayed in the Quantity field after the order is acknowledged. This method increases or decreases an already existing position. Positive percents will increase a position, negative percents will decrease a position. For exmaple, to fully close out a position, you just need to specify percentage to be -100.

BUY ORDER	*Positive Percent*	*Negative Percent*
Long Position	Increases position	No effect
Short Position	No effect	Decreases position
SELL ORDER	*Positive Percent*	*Negative Percent*
Long Position	No effect	Decreases position
Short Position	Increases position	No effect

Allocation by Profile

Percentages

This method will split the total number of shares in the order between listed accounts based on the percentages you indicate. For example, an order for 1000 shares using a profile with four accounts at 25% each would allocate 250 shares to each listed account in the profile.

Financial Ratios

This method calculates the allocation of shares based on the ratios you enter. For example, an order for 1000 shares using a profile with four accounts set to a ratio of 4, 2, 1, 1 would allocate 500, 250, 125 and 125 shares to the listed accounts, respectively.

Shares

This method allocates the absolute number of shares you enter to each account listed. If you use this method, the order size is calculated by adding together the number of shares allocated to each account in the profile.

Table 15.2 Profile Methods Matching

Profile Methods	Type Number
Percentages	Type - 1
Financial Ratios	Type - 2
Shares	Type - 3

15.2 Groups and Profiles from the API

The **IBApi.EClient.requestFA** function allows Financial Advisor to manually request current allocation configuration data from TWS.

Request Account Aliases

- ```
  client.requestFA(Constants.FaAliases);
  ```
- ```
  client.requestFA(FADataType.ALIASES.ordinal());
  ```
- ```
  client.requestFA(Constants.FaAliases)
  ```
- ```
  m_pClient->requestFA(faDataType::ALIASES);
  ```
- ```
  self.requestFA(FaDataTypeEnum.ALIASES)
  ```

Request FA Groups

- ```
  client.requestFA(Constants.FaGroups);
  ```
- ```
  client.requestFA(FADataType.GROUPS.ordinal());
  ```
- ```
  client.requestFA(Constants.FaGroups)
  ```
- ```
  m_pClient->requestFA(faDataType::GROUPS);
  ```
- ```
  self.requestFA(FaDataTypeEnum.GROUPS)
  ```

Request FA Profiles

- ```
  client.requestFA(Constants.FaProfiles);
  ```
- ```
  client.requestFA(FADataType.PROFILES.ordinal());
  ```
- ```
  client.requestFA(Constants.FaProfiles)
  ```
- ```
  m_pClient->requestFA(faDataType::PROFILES);
  ```
- ```
  self.requestFA(FaDataTypeEnum.PROFILES)
  ```

The resulting FA allocation configuration will be delivered via the **IBApi.EWrapper.receiveFA** (p. **??**). The event includes an XML string containing the requested information.

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
      public virtual void receiveFA(int faDataType, string faXmlData)
      {
          Console.WriteLine("Receing FA: "+faDataType+" - "+faXmlData);
      }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void receiveFA(int faDataType, String xml) {
          System.out.println("Receiving FA: "+faDataType+" - "+xml);
      }
  ```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

  ...

```
        Public Sub receiveFA(faDataType As Integer, faXmlData As String) Implements
       IBApi.EWrapper.receiveFA
            Console.WriteLine("Receing FA: " & faDataType & " - " & faXmlData)
        End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::receiveFA(faDataType pFaDataType, const std::string& cxml) {
      std::cout << "Receiving FA: " << (int)pFaDataType << std::endl << cxml << std::endl;
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def receiveFA(self, faData: FaDataType, cxml: str):
          super().receiveFA(faData, cxml)
          print("Receiving FA: ", faData)
          open('log/fa.xml', 'w').write(cxml)
  ```

The **IBApi.EClient.replaceFA** function can be called to replace the previous FA configuration in TWS by passing in a FULL XML string that contains all allocation information.

Replace Account Groups Configuration

- ```
          public static string FaOneGroup = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
              + "<ListOfGroups>"
              + "<Group>"
                  + "<name>Equal_Quantity</name>"
                  + "<ListOfAccts varName=\"list\">"
                      //Replace with your own accountIds
                      + "<String>DU119915</String>"
                      + "<String>DU119916</String>"
                  + "</ListOfAccts>"
                  + "<defaultMethod>EqualQuantity</defaultMethod>"
              + "</Group>"
          + "</ListOfGroups>";
  ```

  ...

  ```
              client.replaceFA(Constants.FaGroups, FaAllocationSamples.FaOneGroup);
  ```

- ```
      public static final String FA_ONE_GROUP = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
              + "<ListOfGroups>"
              + "<Group>"
                  + "<name>Equal_Quantity</name>"
                  + "<ListOfAccts varName=\"list\">"
                      //Replace with your own accountIds
                      + "<String>DU119915</String>"
                      + "<String>DU119916</String>"
                  + "</ListOfAccts>"
                  + "<defaultMethod>EqualQuantity</defaultMethod>"
              + "</Group>"
          + "</ListOfGroups>";
  ```

  ...

  ```
          client.replaceFA(FADataType.GROUPS.ordinal(), FAMethodSamples.FA_ONE_GROUP);
  ```

- ```
          'Replace with your own accountIds
          Public Shared FaOneGroup As String = "<?xml version=""1.0"" encoding=""UTF-8""?>" &
                 "<ListOfGroups>" &
                 "<Group>" &
                     "<name> Equal_Quantity</name>" &
                     "<ListOfAccts varName=""list"">" &
                          "<String>DU119915</String>" &
                          "<String>DU119916</String>" &
                     "</ListOfAccts>" &
                     "<defaultMethod>EqualQuantity</defaultMethod>" &
                  "</Group>" &
                 "</ListOfGroups>"
  ```

  ...
  ```

```
        client.replaceFA(Constants.FaGroups, FaAllocationSamples.FaOneGroup)
```

-
```
        std::string groupXml =
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
        "<ListOfGroups>"
            "<Group>"
                "<name>Equal_Quantity</name>"
                "<ListOfAccts varName=\"list\">"
                    "<String>DU119915</String>"
                    "<String>DU119916</String>"
                "</ListOfAccts>"
                "<defaultMethod>EqualQuantity</defaultMethod>"
            "</Group>"
        "</ListOfGroups>";
```

...

```
    m_pClient->replaceFA(faDataType::GROUPS, FAMethodSamples::FAOneGroup());
```

-
```
    FaOneGroup = "".join(("<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
        , "<ListOfGroups>"
                    , "<Group>"
                            , "<name>Equal_Quantity</name>"
                            , "<ListOfAccts varName=\"list\">"
                                    #Replace with your own accountIds
                                    , "<String>DU119915</String>"
                                    , "<String>DU119916</String>"
                            , "</ListOfAccts>"
                            , "<defaultMethod>EqualQuantity</defaultMethod>"
                    , "</Group>"
                , "</ListOfGroups>"))
```

...

```
        self.replaceFA(FaDataTypeEnum.GROUPS, FaAllocationSamples.FaOneGroup)
```

Note: *The above command will replace any previous FA group configuration in TWS with one 'Group_Equal_↩ Quantity' group allocation.*

-
```
        public static string FaTwoGroups = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
            +"<ListOfGroups>"
            + "<Group>"
                + "<name>Equal_Quantity</name>"
                + "<ListOfAccts varName=\"list\">"
                    //Replace with your own accountIds
                    + "<String>DU119915</String>"
                    + "<String>DU119916</String>"
                + "</ListOfAccts>"
                + "<defaultMethod>EqualQuantity</defaultMethod>"
            + "</Group>"
            + "<Group>"
                + "<name>Pct_Change</name>"
                + "<ListOfAccts varName=\"list\">"
                    //Replace with your own accountIds
                    + "<String>DU119915</String>"
                    + "<String>DU119916</String>"
                + "</ListOfAccts>"
                + "<defaultMethod>PctChange</defaultMethod>"
            + "</Group>"
        + "</ListOfGroups>";
```

...

```
            client.replaceFA(Constants.FaGroups, FaAllocationSamples.FaTwoGroups);
```

-
```
    public static final String FA_TWO_GROUPS = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
            +"<ListOfGroups>"
            + "<Group>"
                + "<name>Equal_Quantity</name>"
                + "<ListOfAccts varName=\"list\">"
                    //Replace with your own accountIds
                    + "<String>DU119915</String>"
                    + "<String>DU119916</String>"
                + "</ListOfAccts>"
                + "<defaultMethod>EqualQuantity</defaultMethod>"
            + "</Group>"
            + "<Group>"
                + "<name>Pct_Change</name>"
                + "<ListOfAccts varName=\"list\">"
```

```
                    //Replace with your own accountIds
                    + "<String>DU119915</String>"
                    + "<String>DU119916</String>"
                + "</ListOfAccts>"
                + "<defaultMethod>PctChange</defaultMethod>"
            + "</Group>"
        + "</ListOfGroups>";
```

...

```
client.replaceFA(FADataType.GROUPS.ordinal(), FAMethodSamples.FA_TWO_GROUPS);
```

- ```
  'Replace with your own accountIds
  Public Shared FaTwoGroups As String = "<?xml version=""1.0"" encoding=""UTF-8""?>" &
  "<ListOfGroups>" &
  "<Group>" &
  "<name> Equal_Quantity</name>" &
           "<ListOfAccts varName = ""list"">" &
               "<String> DU119915</String>" &
               "<String> DU119916</String>" &
           "</ListOfAccts>" &
           "<defaultMethod> EqualQuantity</defaultMethod>" &
       "</Group>" &
       "<Group>" &
  "<name> Pct_Change</name>" &
           "<ListOfAccts varName = ""list"">" &
               "<String> DU119915</String>" &
               "<String> DU119916</String>" &
           "</ListOfAccts>" &
           "<defaultMethod> PctChange</defaultMethod>" &
       "</Group>" &
   "</ListOfGroups>"
  ```

...

```
client.replaceFA(Constants.FaGroups, FaAllocationSamples.FaTwoGroups)
```

- ```
  std::string groupXml =
  "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
  "<ListOfGroups>"
      "<Group>"
          "<name>Equal_Quantity</name>"
          "<ListOfAccts varName=\"list\">"
              "<String>DU119915</String>"
              "<String>DU119916</String>"
          "</ListOfAccts>"
          "<defaultMethod>EqualQuantity</defaultMethod>"
      "</Group>"
      "<Group>"
          "<name>Pct_Change</name>"
          "<ListOfAccts varName=\"list\">"
              "<String>DU119915</String>"
              "<String>DU119916</String>"
          "</ListOfAccts>"
          "<defaultMethod>PctChange</defaultMethod>"
      "</Group>"
  "</ListOfGroups>";
  ```

...

```
m_pClient->replaceFA(faDataType::GROUPS, FAMethodSamples::FATwoGroups());
```

- ```
  FaTwoGroups = "".join(("<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
              ,"<ListOfGroups>"
                  , "<Group>"
                          , "<name>Equal_Quantity</name>"
                          , "<ListOfAccts varName=\"list\">"
                                  #Replace with your own accountIds
                                  , "<String>DU119915</String>"
                                  , "<String>DU119916</String>"
                          , "</ListOfAccts>"
                          , "<defaultMethod>EqualQuantity</defaultMethod>"
                  , "</Group>"
                  , "<Group>"
                          , "<name>Pct_Change</name>"
                          , "<ListOfAccts varName=\"list\">"
              #Replace with your own accountIds
              , "<String>DU119915</String>"
                          , "<String>DU119916</String>"
                          , "</ListOfAccts>"
                          , "<defaultMethod>PctChange</defaultMethod>"
                  , "</Group>"
          , "</ListOfGroups>"))
  ```

...
  ```

```
self.replaceFA(FaDataTypeEnum.GROUPS, FaAllocationSamples.FaTwoGroups)
```

Note: *The above command will add another 'Group_Pct_Change' group allocation to the previous FA configuration.*

Replace Account Profiles Configuration

You can find the corresponding type number for profile allocation in the "Profile Methods Matching" table above.

- ```
  public static string FaOneProfile = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
      + "<ListOfAllocationProfiles>"
      + "<AllocationProfile>"
          + "<name>Percent_60_40</name>"
          + "<type>1</type>"
          + "<ListOfAllocations varName=\"listOfAllocations\">"
              + "<Allocation>"
                  //Replace with your own accountIds
                  + "<acct>DU119915</acct>"
                  + "<amount>60.0</amount>"
              + "</Allocation>"
              + "<Allocation>"
                  //Replace with your own accountIds
                  + "<acct>DU119916</acct>"
                  + "<amount>40.0</amount>"
              + "</Allocation>"
          + "</ListOfAllocations>"
      + "</AllocationProfile>"
  + "</ListOfAllocationProfiles>";
  ```

  ...

  ```
  client.replaceFA(Constants.FaProfiles, FaAllocationSamples.FaOneProfile);
  ```

- ```
  public static final String FA_ONE_PROFILE = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
          + "<ListOfAllocationProfiles>"
          + "<AllocationProfile>"
              + "<name>Percent_60_40</name>"
              + "<type>1</type>"
              + "<ListOfAllocations varName=\"listOfAllocations\">"
                  + "<Allocation>"
                      //Replace with your own accountIds
                      + "<acct>DU119915</acct>"
                      + "<amount>60.0</amount>"
                  + "</Allocation>"
                  + "<Allocation>"
                      //Replace with your own accountIds
                      + "<acct>DU119916</acct>"
                      + "<amount>40.0</amount>"
                  + "</Allocation>"
              + "</ListOfAllocations>"
          + "</AllocationProfile>"
      + "</ListOfAllocationProfiles>";
  ```

  ...

  ```
  client.replaceFA(FADataType.PROFILES.ordinal(), FAMethodSamples.FA_ONE_PROFILE);
  ```

- ```
  'Replace with your own accountIds
  Public Shared FaOneProfile As String = "<?xml version=""1.0"" encoding=""UTF-8""?>" &
      "<ListOfAllocationProfiles>" &
      "<AllocationProfile>" &
          "<name> Percent_60_40</name>" &
          "<Type>1</type>" &
              "<ListOfAllocations varName=""listOfAllocations"">" &
              "<Allocation>" &
               "<acct> DU119915</acct>" &
               "<amount>60.0</amount>" &
              "</Allocation>" &
             "<Allocation>" &
              "<acct> DU119916</acct>" &
               "<amount>40.0</amount>" &
             "</Allocation>" &
              "</ListOfAllocations>" &
     "</AllocationProfile>" &
    "</ListOfAllocationProfiles>"
  ```

  ...

  ```
  client.replaceFA(Constants.FaProfiles, FaAllocationSamples.FaOneProfile)
  ```
  ```

-
```
        std::string profileXml =
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
        "<ListOfAllocationProfiles>"
            "<AllocationProfile>"
                "<name>Percent_60_40</name>"
                "<type>1</type>"
                "<ListOfAllocations varName=\"listOfAllocations\">"
                    "<Allocation>"
                        "<acct>DU119915</acct>"
                        "<amount>60.0</amount>"
                    "</Allocation>"
                    "<Allocation>"
                        "<acct>DU119916</acct>"
                        "<amount>40.0</amount>"
                    "</Allocation>"
                "</ListOfAllocations>"
            "</AllocationProfile>"
        "</ListOfAllocationProfiles>";
```

...

```
    m_pClient->replaceFA(faDataType::PROFILES, FAMethodSamples::FAOneProfile());
```

-
```
    FaOneProfile = "".join(("<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
                , "<ListOfAllocationProfiles>"
                    , "<AllocationProfile>"
                            , "<name>Percent_60_40</name>"
                            , "<type>1</type>"
                            , "<ListOfAllocations varName=\"listOfAllocations\">"
                                    , "<Allocation>"
                                            #Replace with your own accountIds
                                            , "<acct>DU119915</acct>"
                                            , "<amount>60.0</amount>"
                                    , "</Allocation>"
                                    , "<Allocation>"
                                            #Replace with your own accountIds
                                            , "<acct>DU119916</acct>"
                                            , "<amount>40.0</amount>"
                                    , "</Allocation>"
                            , "</ListOfAllocations>"
                    , "</AllocationProfile>"
                , "</ListOfAllocationProfiles>"))
```

...

```
        self.replaceFA(FaDataTypeEnum.PROFILES, FaAllocationSamples.FaOneProfile)
```

Note: *The above command will replace any previous FA profile configuration in TWS with one 'Percent_60_40' profile allocation*

-
```
        public static string FaTwoProfiles = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
            + "<ListOfAllocationProfiles>"
            + "<AllocationProfile>"
                + "<name>Percent_60_40</name>"
                + "<type>1</type>"
                + "<ListOfAllocations varName=\"listOfAllocations\">"
                    + "<Allocation>"
                        //Replace with your own accountIds
                        + "<acct>DU119915</acct>"
                        + "<amount>60.0</amount>"
                    + "</Allocation>"
                    + "<Allocation>"
                        //Replace with your own accountIds
                        + "<acct>DU119916</acct>"
                        + "<amount>40.0</amount>"
                    + "</Allocation>"
                + "</ListOfAllocations>"
            + "</AllocationProfile>"
            + "<AllocationProfile>"
                + "<name>Ratios_2_1</name>"
                + "<type>1</type>"
                + "<ListOfAllocations varName=\"listOfAllocations\">"
                    + "<Allocation>"
                        //Replace with your own accountIds
                        + "<acct>DU119915</acct>"
                        + "<amount>2.0</amount>"
                    + "</Allocation>"
                    + "<Allocation>"
                        //Replace with your own accountIds
                        + "<acct>DU119916</acct>"
                        + "<amount>1.0</amount>"
```

```
                    + "</Allocation>"
                + "</ListOfAllocations>"
            + "</AllocationProfile>"
        + "</ListOfAllocationProfiles>";
```

...

```
            client.replaceFA(Constants.FaProfiles, FaAllocationSamples.FaTwoProfiles);
```

- ```
    public static final String FA_TWO_PROFILES = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
            + "<ListOfAllocationProfiles>"
            + "<AllocationProfile>"
                + "<name>Percent_60_40</name>"
                + "<type>1</type>"
                + "<ListOfAllocations varName=\"listOfAllocations\">"
                    + "<Allocation>"
                        //Replace with your own accountIds
                        + "<acct>DU119915</acct>"
                        + "<amount>60.0</amount>"
                    + "</Allocation>"
                    + "<Allocation>"
                        //Replace with your own accountIds
                        + "<acct>DU119916</acct>"
                        + "<amount>40.0</amount>"
                    + "</Allocation>"
                + "</ListOfAllocations>"
            + "</AllocationProfile>"
            + "<AllocationProfile>"
                + "<name>Ratios_2_1</name>"
                + "<type>1</type>"
                + "<ListOfAllocations varName=\"listOfAllocations\">"
                    + "<Allocation>"
                        //Replace with your own accountIds
                        + "<acct>DU119915</acct>"
                        + "<amount>2.0</amount>"
                    + "</Allocation>"
                    + "<Allocation>"
                        //Replace with your own accountIds
                        + "<acct>DU119916</acct>"
                        + "<amount>1.0</amount>"
                    + "</Allocation>"
                + "</ListOfAllocations>"
            + "</AllocationProfile>"
        + "</ListOfAllocationProfiles>";
    ```

...

```
        client.replaceFA(FADataType.PROFILES.ordinal(), FAMethodSamples.FA_TWO_PROFILES);
```

- ```
    'Replace with your own accountIds
    Public Shared FaTwoProfiles As String = "<?xml version=""1.0"" encoding=""UTF-8""?>" &
         "<ListOfAllocationProfiles>" &
    "<AllocationProfile>" &
    "<name> Percent_60_40</name>" &
             "<Type>1</type>" &
             "<ListOfAllocations varName = ""listOfAllocations""">" &
                 "<Allocation>" &
    "<acct> DU119915</acct>" &
                     "<amount>60.0</amount>" &
                 "</Allocation>" &
                 "<Allocation>" &
    "<acct> DU119916</acct>" &
                     "<amount>40.0</amount>" &
                 "</Allocation>" &
             "</ListOfAllocations>" &
         "</AllocationProfile>" &
         "<AllocationProfile>" &
    "<name> Ratios_2_1</name>" &
             "<Type>1</Type>" &
             "<ListOfAllocations varName = ""listOfAllocations""">" &
                 "<Allocation>" &
                     "<acct> DU119915</acct>" &
                     "<amount>2.0</amount>" &
                 "</Allocation>" &
                 "<Allocation>" &
    "<acct> DU119916</acct>" &
                     "<amount>1.0</amount>" &
                 "</Allocation>" &
             "</ListOfAllocations>" &
         "</AllocationProfile>" &
     "</ListOfAllocationProfiles>"
    ```

...

```
        client.replaceFA(Constants.FaProfiles, FaAllocationSamples.FaTwoProfiles)
```

- ```
        std::string profileXml =
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
        "<ListOfAllocationProfiles>"
            "<AllocationProfile>"
                "<name>Percent_60_40</name>"
                "<type>1</type>"
                "<ListOfAllocations varName=\"listOfAllocations\">"
                    "<Allocation>"
                        "<acct>DU119915</acct>"
                        "<amount>60.0</amount>"
                    "</Allocation>"
                    "<Allocation>"
                        "<acct>DU119916</acct>"
                        "<amount>40.0</amount>"
                    "</Allocation>"
                "</ListOfAllocations>"
            "</AllocationProfile>"
            "<AllocationProfile>"
                "<name>Ratios_2_1</name>"
                "<type>1</type>"
                "<ListOfAllocations varName=\"listOfAllocations\">"
                    "<Allocation>"
                        "<acct>DU119915</acct>"
                        "<amount>2.0</amount>"
                    "</Allocation>"
                    "<Allocation>"
                        "<acct>DU119916</acct>"
                        "<amount>1.0</amount>"
                    "</Allocation>"
                "</ListOfAllocations>"
            "</AllocationProfile>"
        "</ListOfAllocationProfiles>";
  ```

...

```
m_pClient->replaceFA(faDataType::PROFILES, FAMethodSamples::FATwoProfiles());
```

- ```
    FaTwoProfiles = "".join(("<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
                , "<ListOfAllocationProfiles>"
                    , "<AllocationProfile>"
                            , "<name>Percent_60_40</name>"
                            , "<type>1</type>"
                            , "<ListOfAllocations varName=\"listOfAllocations\">"
                                    , "<Allocation>"
                                            #Replace with your own accountIds
                                            , "<acct>DU119915</acct>"
                                            , "<amount>60.0</amount>"
                                    , "</Allocation>"
                                    , "<Allocation>"
                                            #Replace with your own accountIds
                                            , "<acct>DU119916</acct>"
                                            , "<amount>40.0</amount>"
                                    , "</Allocation>"
                            , "</ListOfAllocations>"
                    , "</AllocationProfile>"
                    , "<AllocationProfile>"
                            , "<name>Ratios_2_1</name>"
                            , "<type>1</type>"
                            , "<ListOfAllocations varName=\"listOfAllocations\">"
                                    , "<Allocation>"
                                            #Replace with your own accountIds
                                            , "<acct>DU119915</acct>"
                                            , "<amount>2.0</amount>"
                                    , "</Allocation>"
                                    , "<Allocation>"
                                            #Replace with your own accountIds
                                            , "<acct>DU119916</acct>"
                                            , "<amount>1.0</amount>"
                                    , "</Allocation>"
                            , "</ListOfAllocations>"
                    , "</AllocationProfile>"
                , "</ListOfAllocationProfiles>"))
  ```

...

```
        self.replaceFA(FaDataTypeEnum.PROFILES, FaAllocationSamples.FaTwoProfiles)
```

Note: *The above command will add another 'Group_Pct_Change' group allocation to the previous FA configuration*

Model Portfolios and the API

Placing Orders to a FA account

15.3 Model Portfolios and the API

Advisors can use Model Portfolios to easily invest some or all of a client's assets into one or multiple custom-created portfolios, rather than tediously managing individual investments in single instruments.

More about Model Portfolios

The TWS API can access model portfolios in accounts where this functionality is available and a specific model has previously been setup in TWS. API functionality allows the client application to request model position update subscriptions, request model account update subscriptions, or place orders to a specific model.

Model Portfolio functionality **not** available in the TWS API:

- Portfolio Model Creation
- Portfolio Model Rebalancing
- Portfolio Model Position or Cash Transfer

To request position updates from a specific model, the function **IBApi::EClient::reqPositionsMulti** can be used: **Position Update Subscription by Model**

To request model account updates, there is the function **IBApi::EClient::reqAccountUpdatesMulti** (p. **??**), see: **Account Value Update Subscriptions by Model**

To place an order to a model, the **IBApi.Order.ModelCode** field must be set accordingly, for example:

- ```
  Order modelOrder = OrderSamples.LimitOrder("BUY", 200, 100);
  modelOrder.Account = "DF12345";  // master FA account number
  modelOrder.ModelCode = "Technology"; // model for tech stocks first created in TWS
  client.placeOrder(nextOrderId++, ContractSamples.USStock(), modelOrder);
  ```

- ```
  Order modelOrder = OrderSamples.LimitOrder("BUY", 200, 100);
  modelOrder.account("DF12345");  // master FA account number
  modelOrder.modelCode("Technology"); // model for tech stocks first created in TWS
  client.placeOrder(nextOrderId++, ContractSamples.USStock(), modelOrder);
  ```

- ```
  Dim modelOrder As Order = OrderSamples.LimitOrder("BUY", 200, 100)
  modelOrder.Account = "DF12345" 'master FA account number
  modelOrder.ModelCode = "Technology" 'model for tech stocks first created in TWS
  client.placeOrder(increment(nextOrderId), ContractSamples.USStock(), modelOrder)
  ```

- ```
  Order modelOrder = OrderSamples::LimitOrder("BUY", 200, 100);
  modelOrder.account = "DF12345";
  modelOrder.modelCode = "Technology";
  m_pClient->placeOrder(m_orderId++, ContractSamples::USStock(), modelOrder);
  ```

- ```
  modelOrder = OrderSamples.LimitOrder("BUY", 200, 100)
  modelOrder.account = "DF12345"
  modelOrder.modelCode = "Technology" # model for tech stocks first created in TWS
  self.placeOrder(self.nextOrderId(), ContractSamples.USStock(), modelOrder)
  ```
  ```

15.4 Placing Orders to a FA account

15.4.1 Account Allocations

Financial Advisor users can invoke **IBApi.EClient.placeOrder** function while specifying corresponding Fi-nancial Advisor Fields in the **IBApi.Order** object. **Important**: Unlike in TWS, there is not a default account allocation for the API- it must be specified with every order placed.

Place an Order for a Single Managed Account

In linked accounts (not Financial Advisor accounts) in particular, it will be necessary to specify an account number in the IBApi::Order::account field with every order placed.

- ```
            Order faOrderOneAccount = OrderSamples.MarketOrder("BUY", 100);
            // Specify the Account Number directly
            faOrderOneAccount.Account = "DU119915";
            client.placeOrder(nextOrderId++, ContractSamples.USStock(), faOrderOneAccount);
  ```

- ```
        Order faOrderOneAccount = OrderSamples.MarketOrder("BUY", 100);
        // Specify the Account Number directly
        faOrderOneAccount.account("DU119915");
        client.placeOrder(nextOrderId++, ContractSamples.USStock(), faOrderOneAccount);
  ```

- ```
        Dim faOrderOneAccount As Order = OrderSamples.MarketOrder("BUY", 100)
        ' Specify the Account Number directly
        faOrderOneAccount.Account = "DU119915"
        client.placeOrder(increment(nextOrderId), ContractSamples.USStock(), faOrderOneAccount)
  ```

- ```
    Order faOrderOneAccount = OrderSamples::MarketOrder("BUY", 100);
    // Specify the Account Number directly
    faOrderOneAccount.account = "DU119915";
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStock(), faOrderOneAccount);
  ```

- ```
        faOrderOneAccount = OrderSamples.MarketOrder("BUY", 100)
        # Specify the Account Number directly
        faOrderOneAccount.account = "DU119915"
        self.placeOrder(self.nextOrderId(), ContractSamples.USStock(), faOrderOneAccount)
  ```

Place an Order for an Account Group

For *EqualQuantity*, *NetLiq* and *AvailableEquity* allocation methods, you need to specify the **IBApi.Order.FaGroup** and **IBApi.Order.FaMethod** parameters.

- ```
            Order faOrderGroupEQ = OrderSamples.LimitOrder("SELL", 200, 2000);
            faOrderGroupEQ.FaGroup = "Group_Equal_Quantity";
            faOrderGroupEQ.FaMethod = "EqualQuantity";
            client.placeOrder(nextOrderId++, ContractSamples.SimpleFuture(), faOrderGroupEQ);
  ```

- ```
        Order faOrderGroupEQ = OrderSamples.LimitOrder("SELL", 200, 2000);
        faOrderGroupEQ.faGroup("Group_Equal_Quantity");
        faOrderGroupEQ.faMethod("EqualQuantity");
        client.placeOrder(nextOrderId++, ContractSamples.USStock(), faOrderGroupEQ);
  ```

- ```
        Dim faOrderGroupEQ As Order = OrderSamples.LimitOrder("SELL", 200, 2000)
        faOrderGroupEQ.FaGroup = "Group_Equal_Quantity"
        faOrderGroupEQ.FaMethod = "EqualQuantity"
        client.placeOrder(increment(nextOrderId), ContractSamples.SimpleFuture(), faOrderGroupEQ)
  ```

- ```
    Order faOrderGroupEQ = OrderSamples::LimitOrder("SELL", 200, 2000);
    faOrderGroupEQ.faGroup = "Group_Equal_Quantity";
    faOrderGroupEQ.faMethod = "EqualQuantity";
    m_pClient->placeOrder(m_orderId++, ContractSamples::SimpleFuture(), faOrderGroupEQ);
  ```

- ```
        faOrderGroupEQ = OrderSamples.LimitOrder("SELL", 200, 2000)
        faOrderGroupEQ.faGroup = "Group_Equal_Quantity"
        faOrderGroupEQ.faMethod = "EqualQuantity"
        self.placeOrder(self.nextOrderId(), ContractSamples.SimpleFuture(), faOrderGroupEQ)
  ```

For *PctChange* allocation methods, you should **NOT** specify the order size but you need to specify **IBApi.Order.↩ FaPercentage** parameter.

```
            Order faOrderGroupPC = OrderSamples.MarketOrder("BUY", 0); ;
            // You should not specify any order quantity for PctChange allocation method
            faOrderGroupPC.FaGroup = "Pct_Change";
            faOrderGroupPC.FaMethod = "PctChange";
            faOrderGroupPC.FaPercentage = "100";
            client.placeOrder(nextOrderId++, ContractSamples.EurGbpFx(), faOrderGroupPC);
```

```
        Order faOrderGroupPC = OrderSamples.MarketOrder("BUY", 0);
        // You should not specify any order quantity for PctChange allocation method
        faOrderGroupPC.faGroup("Pct_Change");
        faOrderGroupPC.faMethod("PctChange");
        faOrderGroupPC.faPercentage("100");
        client.placeOrder(nextOrderId++, ContractSamples.EurGbpFx(), faOrderGroupPC);
```

```
        Dim faOrderGroupPC As Order = OrderSamples.MarketOrder("BUY", 0)
        ' You should not specify any order quantity for PctChange allocation method
        faOrderGroupPC.FaGroup = "Pct_Change"
        faOrderGroupPC.FaMethod = "PctChange"
        faOrderGroupPC.FaPercentage = "100"
        client.placeOrder(increment(nextOrderId), ContractSamples.EurGbpFx(), faOrderGroupPC)
```

```
    Order faOrderGroupPC;
    faOrderGroupPC.action = "BUY";
    faOrderGroupPC.orderType = "MKT";
    // You should not specify any order quantity for PctChange allocation method
    faOrderGroupPC.faGroup = "Pct_Change";
    faOrderGroupPC.faMethod = "PctChange";
    faOrderGroupPC.faPercentage = "100";
    m_pClient->placeOrder(m_orderId++, ContractSamples::EurGbpFx(), faOrderGroupPC);
```

```
        faOrderGroupPC = OrderSamples.MarketOrder("BUY", 0)
        # You should not specify any order quantity for PctChange allocation method
        faOrderGroupPC.faGroup = "Pct_Change"
        faOrderGroupPC.faMethod = "PctChange"
        faOrderGroupPC.faPercentage = "100"
        self.placeOrder(self.nextOrderId(), ContractSamples.EurGbpFx(), faOrderGroupPC)
```

Place an Order for an Account Profile

```
            Order faOrderProfile = OrderSamples.LimitOrder("BUY", 200, 100);
            faOrderProfile.FaProfile = "Percent_60_40";
            client.placeOrder(nextOrderId++, ContractSamples.EuropeanStock(), faOrderProfile);
```

```
        Order faOrderProfile = OrderSamples.LimitOrder("BUY", 200, 100);
        faOrderProfile.faProfile("Percent_60_40");
        client.placeOrder(nextOrderId++, ContractSamples.EuropeanStock(), faOrderProfile);
```

```
        Dim faOrderProfile As Order = OrderSamples.LimitOrder("BUY", 200, 100)
        faOrderProfile.FaProfile = "Percent_60_40"
        client.placeOrder(increment(nextOrderId), ContractSamples.EuropeanStock(), faOrderProfile)
```

```
    Order faOrderProfile = OrderSamples::LimitOrder("BUY", 200, 100);
    faOrderProfile.faProfile = "Percent_60_40";
    m_pClient->placeOrder(m_orderId++, ContractSamples::EuropeanStock(), faOrderProfile);
```

```
        faOrderProfile = OrderSamples.LimitOrder("BUY", 200, 100)
        faOrderProfile.faProfile = "Percent_60_40"
        self.placeOrder(self.nextOrderId(), ContractSamples.EuropeanStock(), faOrderProfile)
```

15.4.2 Model Allocations

```
            Order modelOrder = OrderSamples.LimitOrder("BUY", 200, 100);
            modelOrder.Account = "DF12345";  // master FA account number
            modelOrder.ModelCode = "Technology"; // model for tech stocks first created in TWS
            client.placeOrder(nextOrderId++, ContractSamples.USStock(), modelOrder);
```

```
        Order modelOrder = OrderSamples.LimitOrder("BUY", 200, 100);
        modelOrder.account("DF12345");  // master FA account number
        modelOrder.modelCode("Technology"); // model for tech stocks first created in TWS
        client.placeOrder(nextOrderId++, ContractSamples.USStock(), modelOrder);
```

```
        Dim modelOrder As Order = OrderSamples.LimitOrder("BUY", 200, 100)
        modelOrder.Account = "DF12345" 'master FA account number
        modelOrder.ModelCode = "Technology" 'model for tech stocks first created in TWS
        client.placeOrder(increment(nextOrderId), ContractSamples.USStock(), modelOrder)
```

```
    Order modelOrder = OrderSamples::LimitOrder("BUY", 200, 100);
    modelOrder.account = "DF12345";
    modelOrder.modelCode = "Technology";
    m_pClient->placeOrder(m_orderId++, ContractSamples::USStock(), modelOrder);
```

```
        modelOrder = OrderSamples.LimitOrder("BUY", 200, 100)
        modelOrder.account = "DF12345"
        modelOrder.modelCode = "Technology" # model for tech stocks first created in TWS
        self.placeOrder(self.nextOrderId(), ContractSamples.USStock(), modelOrder)
```

15.4.3 Soft Dollar Tiers

Starting from API version 9.72.18 or higher and TWS Build 959 or higher, Financial Advisors, Hedge Fund and Mutual Fund are able to utilize the Soft Dollar Tiers feature via the API.

Request Soft Dollar Tiers

The **IBApi.EClient.reqSoftDollarTiers** function can be called to manually request current Soft Dollar Tiers structure from TWS.

```
            client.reqSoftDollarTiers(4001);
```

```
                client.reqSoftDollarTiers(4001);
```

```
        client.reqSoftDollarTiers(4001)
```

```
    m_pClient->reqSoftDollarTiers(4001);
```

```
        self.reqSoftDollarTiers(14001)
```

Receive Soft Dollar Tiers

The **IBApi.EWrapper.softDollarTiers** call back function is triggered for returning Soft Dollar Tiers Value, Name and Display Name.

```
    public class EWrapperImpl : EWrapper
    {
```

...

```
        public void softDollarTiers(int reqId, SoftDollarTier[] tiers)
        {
            Console.WriteLine("Soft Dollar Tiers:");

            foreach (var tier in tiers)
            {
                Console.WriteLine(tier.DisplayName);
            }
        }
```

```
public class EWrapperImpl implements EWrapper {
```

...

```
    @Override
    public void softDollarTiers(int reqId, SoftDollarTier[] tiers) {
        for (SoftDollarTier tier : tiers) {
            System.out.print("tier: " + tier.toString() + ", ");
        }

        System.out.println();
    }
```

```
    Public Class EWrapperImpl
        Implements EWrapper
```

...

```
        Public Sub softDollarTiers(reqid As Integer, tiers As SoftDollarTier()) Implements
       EWrapper.softDollarTiers
            Console.WriteLine("Soft Dollar Tiers:")

            For Each tier In tiers
                Console.WriteLine(tier.DisplayName)
            Next
        End Sub
```

```
class TestCppClient : public EWrapper
{
```

...

```
void TestCppClient::softDollarTiers(int reqId, const std::vector<SoftDollarTier> &tiers) {
    printf("Soft dollar tiers (%lu):", tiers.size());

    for (unsigned int i = 0; i < tiers.size(); i++) {
        printf("%s\n", tiers[i].displayName().c_str());
    }
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def softDollarTiers(self, reqId: int, tiers: list):
          super().softDollarTiers(reqId, tiers)
          print("Soft Dollar Tiers:", tiers)
  ```
  ```

Chapter 15

Fundamental Data

http://interactivebrokers.github.io/tws-api/reuters_fundamentals.html

Reuters Fundamental data for stocks and the **Wall Street Events Horizon Calendar** can be accessed via the TWS API through the **IBApi.EClient.reqFundamentalData** (p. **??**). For the "CalendarReport" it is necessary to have the Wall Street Horizon subscription activated first in `Account Management`. The other report types require the Reuters Fundamentals subscription but this is freely available and will be activated in most accounts by default.

```
            client.reqFundamentalData(8001, ContractSamples.USStock(), "ReportsFinSummary", null);
```

```
        client.reqFundamentalData(8001, ContractSamples.USStock(), "ReportsFinSummary", null);
```

```
        client.reqFundamentalData(8001, ContractSamples.USStock(), "ReportsFinSummary", Nothing)
```

```
    m_pClient->reqFundamentalData(8001, ContractSamples::USStock(), "ReportsFinSummary", TagValueListSPtr()
      );
```

```
        self.reqFundamentalData(8001, ContractSamples.USStock(),
                                "ReportsFinSummary", [])
```

Note how a string attribute specifies the requested report type. In the example above the request was made for the financial summary. See **Report Types** at the bottom of this page for the rest report types available.

Results are delivered via **IBApi.EWrapper.fundamentalData** in form of an XML report:

```
    public class EWrapperImpl : EWrapper
    {
```

...

```
        public virtual void fundamentalData(int reqId, string data)
        {
            Console.WriteLine("FundamentalData. " + reqId + "" + data+"\n");
        }
```

```
public class EWrapperImpl implements EWrapper {
```

...

```
    @Override
    public void fundamentalData(int reqId, String data) {
        System.out.println("FundamentalData. ReqId: ["+reqId+"] - Data: ["+data+"]");
    }
```

```
    Public Class EWrapperImpl
        Implements EWrapper
```

...

```
        Public Sub fundamentalData(reqId As Integer, data As String) Implements
       IBApi.EWrapper.fundamentalData
            Console.WriteLine("FundamentalData - ReqId [" & reqId & "] Data [" & data & "]")
        End Sub
```

- ```
  class TestCppClient : public EWrapper
  {

  ...

  void TestCppClient::fundamentalData(TickerId reqId, const std::string& data) {
      printf( "FundamentalData. ReqId: %ld, %s\n", reqId, data.c_str());
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):

  ...

      def fundamentalData(self, reqId: TickerId, data: str):
          super().fundamentalData(reqId, data)
          print("FundamentalData. ", reqId, data)
  ```

The **IBApi.EClient.reqFundamentalData** request can be cancelled via **IBApi.EClient.cancel↩ FundamentalData** :

- ```
              client.cancelFundamentalData(8001);
  ```
- ```
          client.cancelFundamentalData(8001);
  ```
- ```
          '** Canceling fundamentals request ***
          client.cancelFundamentalData(8001)
  ```
- ```
      m_pClient->cancelFundamentalData(8001);
  ```
- ```
          self.cancelFundamentalData(8001)
  ```

16.1 Report Types

Report Type	Description
ReportsFinSummary	Financial summary
ReportsOwnership	Company's ownership *(Can be large in size)*
ReportSnapshot	Company's financial overview
ReportsFinStatements	Financial Statements
RESC	Analyst Estimates
CalendarReport	Company's calendar

*Fundamental Ratios data can be acquired by requesting generic tick type 258 with **IBApi.EClient.reqMktData** (p. **??**). See both **Available Tick Types** and **Fundamental Ratios** (p. **??**).*
  ```

Chapter 16

Error Handling

http://interactivebrokers.github.io/tws-api/error_handling.html

When a client application sends a message to TWS which requires a response which has an expected response (i.e. placing an order, requesting market data, subscribing to account updates, etc.), TWS will almost either always **1)** respond with the relevant data or **2)** send an error message to **IBApi::EWrapper::error** (p. **??**).

- **Exceptions when no response can occur**: If a market data request is made during a competing session (when a paper user is logged into simultaneously on a different computer from the associated live user) there is not currently a message returned from TWS. Also, if a request is made prior to full establishment of connection (denoted by a returned 2104 or 2106 error code *"Data Server is Ok"*), there may not be a response from the request.

Error messages sent by the TWS are handled by the **IBApi.EWrapper.error** method. The **IBApi.E↩Wrapper.error** event contains the originating request Id (or the orderId in case the error was raised when placing an order), a numeric error code and a brief description. It is important to keep in mind that this function is used for *true* error messages as well as notifications that do not mean anything is wrong.

API Error Messages when TWS is not set to the English Language

- Currently on the Windows platform, error messages are sent using Latin1 encoding. If TWS is launched in a non-Western language, it is recommended to enable the setting at Global Configuration -> API -> Settings to "Show API error messages in English".

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
      public virtual void error(int id, int errorCode, string errorMsg)
      {
          Console.WriteLine("Error. Id: " + id + ", Code: " + errorCode + ", Msg: " + errorMsg + "\n");
      }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void error(int id, int errorCode, String errorMsg) {
          System.out.println("Error. Id: " + id + ", Code: " + errorCode + ", Msg: " + errorMsg + "\n");
      }
  ```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

  ...
  ```

```
        Public Sub [error](id As Integer, errorCode As Integer, errorMsg As String) Implements
       IBApi.EWrapper.error
            Console.WriteLine("Error - Id [" & id & "] ErrorCode [" & errorCode & "] ErrorMsg [" & errorMsg
      & "]")
        End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::error(int id, int errorCode, const std::string& errorString)
  {
      printf( "Error. Id: %d, Code: %d, Msg: %s\n", id, errorCode, errorString.c_str());
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def error(self, reqId: TickerId, errorCode: int, errorString: str):
          super().error(reqId, errorCode, errorString)
          print("Error. Id: ", reqId, " Code: ", errorCode, " Msg: ", errorString)
  ```

For a list containing all available error messages from the TWS, see **Message Codes**

17.1 Message Codes

The TWS uses the **IBApi.EWrapper.error** method not only to deliver errors but also warnings or informative messages. This is done mostly for simplicity's sake. Below is a table with all the messages which can be sent by the TWS/IB Gateway. All messages delivered by the TWS are usually accompanied by a brief but meaningful description pointing in the direction of the problem.

17.1.1 System Message Codes

Remember that the TWS API simply connects to a running TWS/IB Gateway which most of times will be running on your local network if not in the same host as the client application. It is your responsibility to provide reliable connectivity between the TWS and your client application.

The messages in the table below are not a consequence of any action performed by the client application. They are notifications about the connectivity status between the TWS and itself and our servers. Your client application must pay special attention to them and handle the situation accordingly. You are very likely to loose connectivity to our servers at least once a day due to our daily server maintenance downtime as clearly detailed in our `Current System Status` page. Note that after the system reset, the TWS/IB Gateway will automatically reconnect to our servers and you can resume your operations normally.

Important: during a reset period, there may be an interruption in the ability to log in or manage orders. Existing orders (native types) will operate normally although execution reports and simulated orders will be delayed until the reset is complete. It is not recommended to operate during the scheduled reset times.

Code	TWS message	Additional notes
1100	Connectivity between IB and the TWS has been lost.	Your TWS/IB Gateway has been disconnected from IB servers. This can occur because of an internet connectivity issue, a nightly reset of the IB servers, or a competing session.
1101	Connectivity between IB and TWS has been restored- data lost.*	The TWS/IB Gateway has successfully reconnected to IB's servers. Your market data requests have been lost and need to be re-submitted.

Code	TWS message	Additional notes
1102	Connectivity between IB and TWS has been restored- data maintained.	The TWS/IB Gateway has successfully reconnected to IB's servers. Your market data requests have been recovered and there is no need for you to re-submit them.
1300	TWS socket port has been reset and this connection is being dropped. Please reconnect on the new port - <port_num>	The port number in the TWS/IBG settings has been changed during an active API connection.

17.1.2 Warning Message Codes

Code	TWS message	Additional notes
2100	New account data requested from TWS. API client has been unsubscribed from account data.	The TWS only allows one **IBApi.EClient.req↩ AccountUpdates** request at a time. If the client application attempts to subscribe to a sec-ond account without canceling the previous subscription, the new request will override the old one and the TWS will send this message notifying so.
2101	Unable to subscribe to account as the following clients are subscribed to a different account.	If a client application invokes **IBApi.EClient.req↩ AccountUpdates** when there is an active subscription **started by a different client**, the T↩ WS will reject the new subscription request with this message.
2102	Unable to modify this order as it is still being processed.	If you attempt to modify an order before it gets processed by the system, the modification will be rejected. Wait until the order has been fully processed before modifying it. See **Placing Orders** for further details.
2103	A market data farm is disconnected.	Indicates a connectivity problem to an IB server. Outside of the nightly IB server reset, this typically indicates an underlying ISP connectivity issue.
2104	Market data farm connection is OK	A notification that connection to the market data server is ok. This is a notification and **not** a true error condition, and is expected on first establishing connection.
2105	A historical data farm is disconnected.	Indicates a connectivity problem to an IB server. Outside of the nightly IB server reset, this typically indicates an underlying ISP connectivity issue.
2106	A historical data farm is connected.	A notification that connection to the market data server is ok. This is a notification and **not** a true error condition, and is expected on first establishing connection.
2107	A historical data farm connection has become inactive but should be available upon demand.	Whenever a connection to the historical data farm is not being used because there is not an active historical data request, the connection will go inactive in IB Gateway. This does **not** indicate any connectivity issue or problem with IB Gateway. As soon as a historical data request is made the status will change back to active.
2108	A market data farm connection has become inactive but should be available upon demand.	Whenever a connection to our data farms is not needed, it will become dormant. There is nothing abnormal nor wrong with your client application nor with the TWS. You can safely ignore this message.

Code	TWS message	Additional notes
2109	Order Event Warning: Attribute "Outside Regular Trading Hours" is ignored based on the order type and destination. PlaceOrder is now processed.	Indicates the outsideRth flag was set for an order for which there is not a regular vs outside regular trading hour distinction
2110	Connectivity between TWS and server is broken. It will be restored automatically.	Indicates a connectivity problem between TWS or IBG and the IB server. This will usually only occur during the IB nightly server reset; cases at other times indicate a problem in the local ISP connectivity.
2137	Cross Side Warning	This warning message occurs in TWS version 955 and higher. It occurs when an order will change the position in an account from long to short or from short to long. To bypass the warning, a new feature has been added to IB Gateway 956 (or higher) and TWS 957 (or higher) so that once can go to Global Configuration > Messages and disable the "Cross Side Warning".

17.1.3 Client Error Codes

Client errors are those occurring purely on the TWS API client code and as such they are never sent by the TWS. They are mostly errors happening when validating messages before these are sent to the TWS and you are unlikely to receive the vast majority of them. As such only a small list of these errors is documented below. To see all available errors of this type please refer to the **IBApi.EClientErrors** class.

Code	Message	Additional notes
501	Already Connected.	Your client application is already connected to the TWS.
502	Couldn't connect to TWS. Confirm that "Enable ActiveX and Socket Clients" is enabled and connection port is the same as "Socket Port" on the TWS "Edit->Global Configuration...->API->Settings" menu.	When you receive this error message it is either because you have not enabled API connectivity in the TWS and/or you are trying to connect on the wrong port. Refer to the TWS' API Settings as explained in the error message. See also **Connectivity**
503	The TWS is out of date and must be upgraded.	Indicates TWS or IBG is too old for use with the current API version. Can also be triggered if the TWS version does not support a specific API function.
504	Not connected.	You are trying to perform a request without properly connecting and/or after connection to the TWS has been broken probably due to an unhandled exception within your client application.

17.1.4 TWS Error Codes

Code	TWS message	Additional notes
100	Max rate of messages per second has been exceeded.	The client application has exceeded the rate of 50 messages/second. The TWS will likely disconnect the client application after this message.

Code	TWS message	Additional notes
101	Max number of tickers has been reached.	The current number of active market data subscriptions in TWS and the API altogether has been exceeded. This number is calculated based on a formula which is based on the equity, commissions, and quote booster packs in an account. Active lines can be checked in Tws using the Ctrl-Alt-= combination
102	Duplicate ticker ID.	A market data request used a ticker ID which is already in use by an active request.
103	Duplicate order ID.	An order was placed with an order ID that is less than or equal to the order ID of a previous order from this client
104	Can't modify a filled order.	An attempt was made to modify an order which has already been filled by the system.
105	Order being modified does not match original order.	An order was placed with an order ID of a currently open order but basic parameters differed (aside from quantity or price fields)
106	Can't transmit order ID:	
107	Cannot transmit incomplete order.	Order is missing a required field.
109	Price is out of the range defined by the Percentage setting at order defaults frame. The order will not be transmitted.	Price entered is outside the range of prices set in TWS or IB Gateway Order Precautionary Settings
110	The price does not conform to the minimum price variation for this contract.	An entered price field has more digits of precision than is allowed for this particular contract. Minimum increment information can be found on the IB Contracts and Securities Search page.
111	The TIF (Tif type) and the order type are incompatible.	The time in force specified cannot be used with this order type. Please refer to order tickets in TWS for allowable combinations.
113	The Tif option should be set to DAY for MOC and LOC orders.	Market-on-close or Limit-on-close orders should be sent with time in force set to 'DAY'
114	Relative orders are valid for stocks only.	This error is deprecated.
115	Relative orders for US stocks can only be submitted to SMART, SM↩ ART_ECN, INSTINET, or PRIMEX.	This error is deprecated.
116	The order cannot be transmitted to a dead exchange.	Exchange field is invalid.

Code	TWS message	Additional notes
117	The block order size must be at least 50.	
118	VWAP orders must be routed through the VWAP exchange.	
119	Only VWAP orders may be placed on the VWAP exchange.	When an order is routed to the V↩ WAP exchange, the type of the order must be defined as 'VWAP'.
120	It is too late to place a VWAP order for today.	The cutoff has passed for the current day to place VWAP orders.
121	Invalid BD flag for the order. Check "Destination" and "BD" flag.	This error is deprecated.
122	No request tag has been found for order:	
123	No record is available for conid:	The specified contract ID cannot be found. This error is deprecated.
124	No market rule is available for conid:	
125	Buy price must be the same as the best asking price.	
126	Sell price must be the same as the best bidding price.	
129	VWAP orders must be submitted at least three minutes before the start time.	The start time specified in the V↩ WAP order is less than 3 minutes after when it is placed.
131	The sweep-to-fill flag and display size are only valid for US stocks routed through SMART, and will be ignored.	
132	This order cannot be transmitted without a clearing account.	
133	Submit new order failed.	
134	Modify order failed.	
135	Can't find order with ID =	An attempt was made to cancel an order not currently in the system.
136	This order cannot be cancelled.	An attempt was made to cancel an order than cannot be cancelled, for instance because
137	VWAP orders can only be cancelled up to three minutes before the start time.	
138	Could not parse ticker request:	
139	Parsing error:	Error in command syntax generated parsing error.
140	The size value should be an integer:	The size field in the Order class has an invalid type.
141	The price value should be a double:	A price field in the Order type has an invalid type.
142	Institutional customer account does not have account info	
143	Requested ID is not an integer number.	The IDs used in API requests must be integer values.

Code	TWS message	Additional notes
144	Order size does not match total share allocation. To adjust the share allocation, right-click on the order and select â€œModify > Share Allocation.â€	
145	Error in validating entry fields -	An error occurred with the syntax of a request field.
146	Invalid trigger method.	The trigger method specified for a method such as stop or trail stop was not one of the allowable methods.
147	The conditional contract info is incomplete.	
148	A conditional order can only be submitted when the order type is set to limit or market.	This error is deprecated.
151	This order cannot be transmitted without a user name.	In DDE the user name is a required field in the place order command.
152	The "hidden" order attribute may not be specified for this order.	The order in question cannot be placed as a hidden order. See- https://www.↩ interactivebrokers.↩ com/en/index.php?f=596
153	EFPs can only be limit orders.	This error is deprecated.
154	Orders cannot be transmitted for a halted security.	A security was halted for trading when an order was placed.
155	A sizeOp order must have a user name and account.	This error is deprecated.
156	A SizeOp order must go to IBSX	This error is deprecated.
157	An order can be EITHER Iceberg or Discretionary. Please remove either the Discretionary amount or the Display size.	In the Order class extended attributes the fields 'Iceberg' and 'Discretionary' cannot
158	You must specify an offset amount or a percent offset value.	TRAIL and TRAIL STOP orders must have an absolute offset amount or offset percentage specified.
159	The percent offset value must be between 0% and 100%.	A percent offset value was specified outside the allowable range of 0% and 100%.
160	The size value cannot be zero.	The size of an order must be a positive quantity.
161	Cancel attempted when order is not in a cancellable state. Order permId =	An attempt was made to cancel an order not active at the time.
162	Historical market data Service error message.	
163	The price specified would violate the percentage constraint specified in the default order settings.	The order price entered is outside the allowable range specified in the Order Precautionary Settings of TWS or IB Gateway

Code	TWS message	Additional notes
164	There is no market data to check price percent violations.	No market data is available for the specified contract to determine whether the specified price is outside the price percent precautionary order setting.
165	Historical market Data Service query message.	There was an issue with a historical data request, such is no such data in IB's database. Note this message is not specific to the API.
166	HMDS Expired Contract Violation.	Historical data is not available for the specified expired contract.
167	VWAP order time must be in the future.	The start time of a VWAP order has already passed.
168	Discretionary amount does not conform to the minimum price variation for this contract.	The discretionary field is specified with a number of degrees of precision higher than what is allowed for a specified contract.
200	No security definition has been found for the request.	The specified contract does not match any in IB's database, usually because of an incorrect or missing parameter.
	The contract description specified for <Symbol> is ambiguous	Ambiguity may occur when the contract definition provided is not unique. For some stocks that has the same Symbol, Currency and Exchange, you need to specify the **IBApi.Contract.PrimaryExch** attribute to avoid ambiguity. Please refer to a sample stock contract **here** (p. **??**). For futures that has multiple multipliers for the same expiration, You need to specify the **IBApi.Contract.Multiplier** attribute to avoid ambiguity. Please refer to a sample futures contract **here** (p. **??**).
201	Order rejected - Reason:	An attempted order was rejected by the IB servers.
202	Order cancelled - Reason:	An active order on the IB server was cancelled
203	The security <security> is not available or allowed for this account.	The specified security has a trading restriction with a specific account.
300	Can't find EId with ticker Id:	An attempt was made to cancel market data for a ticker ID that was not associated with a current subscription. With the DDE API this occurs by clearing the spreadsheet cell.
301	Invalid ticker action:	
302	Error parsing stop ticker string:	

Code	TWS message	Additional notes
303	Invalid action:	An action field was specified that is not available for the account. For most accounts this is only BUY or SELL. Some institutional accounts also have the options SSHORT or SLONG available.
304	Invalid account value action:	
305	Request parsing error, the request has been ignored.	The syntax of a DDE request is invalid.
306	Error processing DDE request:	An issue with a DDE request prevented it from processing.
307	Invalid request topic:	The 'topic' field in a DDE request is invalid.
308	Unable to create the 'API' page in TWS as the maximum number of pages already exists.	An order placed from the API will automatically open a new page in classic TWS, however there are already the maximum number of pages open.
309	Max number (3) of market depth requests has been reached. Note↩ : TWS currently limits users to a maximum of 3 distinct market depth requests. This same restriction applies to API clients, however A↩ PI clients may make multiple market depth requests for the same security.	
310	Can't find the subscribed market depth with tickerId:	An attempt was made to cancel market depth for a ticker not currently active.
311	The origin is invalid.	The origin field specified in the Order class is invalid.
312	The combo details are invalid.	Combination contract specified has invalid parameters.
313	The combo details for leg '<leg number>' are invalid.	A combo leg was not defined correctly.
314	Security type 'BAG' requires combo leg details.	When specifying security type as 'BAG' make sure to also add combo legs with details.
315	Stock combo legs are restricted to SMART order routing.	Make sure to specify 'SMART' as an exchange when using stock combo contracts.
316	Market depth data has been HA↩ LTED. Please re-subscribe.	You need to re-subscribe to start receiving market depth data again.
317	Market depth data has been RE↩ SET. Please empty deep book contents before applying any new entries.	
319	Invalid log level <log level>	Make sure that you are setting a log level to a value in range of 1 to 5.
320	Server error when reading an API client request.	
321	Server error when validating an API client request.	

Code	TWS message	Additional notes
322	Server error when processing an API client request.	
323	Server error: cause - s	
324	Server error when reading a D↩ DE client request (missing information).	Make sure that you have specified all the needed information for your request.
325	Discretionary orders are not supported for this combination of exchange and order type.	Make sure that you are specifying a valid combination of exchange and order type for the discretionary order.
326	Unable to connect as the client id is already in use. Retry with a unique client id.	Another client application is already connected with the specified client id.
327	Only API connections with clientId set to 0 can set the auto bind TWS orders property.	
328	Trailing stop orders can be attached to limit or stop-limit orders only.	Indicates attempt to attach trail stop to order which was not a limit or stop-limit.
329	Order modify failed. Cannot change to the new order type.	You are not allowed to modify initial order type to the specific order type you are using.
330	Only FA or STL customers can request managed accounts list.	Make sure that your account type is either FA or STL.
331	Internal error. FA or STL does not have any managed accounts.	You do not have any managed accounts.
332	The account codes for the order profile are invalid.	You need to check that the account codes you specified for your request are valid.
333	Invalid share allocation syntax.	
334	Invalid Good Till Date order	Check you order settings.
335	Invalid delta: The delta must be between 0 and 100.	
336	The time or time zone is invalid. The correct format is hh:mm:ss xxx where xxx is an optionally specified time-zone. E.g.: 15:59:00 EST Note that there is a space between the time and the time zone. If no time zone is specified, local time is assumed.	
337	The date, time, or time-zone entered is invalid. The correct format is yyyymmdd hh:mm:ss xxx where yyyymmdd and xxx are optional. E.g.: 20031126 15:59:00 ESTNote that there is a space between the date and time, and between the time and time-zone.	
338	Good After Time orders are currently disabled on this exchange.	
339	Futures spread are no longer supported. Please use combos instead.	

Code	TWS message	Additional notes
340	Invalid improvement amount for box auction strategy.	
341	Invalid delta. Valid values are from 1 to 100. You can set the delta from the "Pegged to Stock" section of the Order Ticket Panel, or by selecting Page/Layout from the main menu and adding the Delta column.	
342	Pegged order is not supported on this exchange.	You can review all order types and supported exchanges on the `Order Types and Algos` page.
343	The date, time, or time-zone entered is invalid. The correct format is yyyymmdd hh:mm:ss xxx	
344	The account logged into is not a financial advisor account.	You are trying to perform an action that is only available for the financial advisor account.
345	Generic combo is not supported for FA advisor account.	
346	Not an institutional account or an away clearing account.	
347	Short sale slot value must be 1 (broker holds shares) or 2 (delivered from elsewhere).	Make sure that your slot value is either 1 or 2.
348	Order not a short sale – type must be SSHORT to specify short sale slot.	Make sure that the action you specified is 'SSHORT'.
349	Generic combo does not support "Good After" attribute.	
350	Minimum quantity is not supported for best combo order.	
351	The "Regular Trading Hours only" flag is not valid for this order.	
352	Short sale slot value of 2 (delivered from elsewhere) requires location.	You need to specify designated↩ Location for your order.
353	Short sale slot value of 1 requires no location be specified.	You do not need to specify designatedLocation for your order.
354	Not subscribed to requested market data.	You do not have live market data available in your account for the specified instruments. For further details please refer to **Streaming Market Data** (p. **??**).
355	Order size does not conform to market rule.	Check order size parameters for the specified contract from the T↩ WS Contract Details.
356	Smart-combo order does not support OCA group.	Remove OCA group from your order.
357	Your client version is out of date.	
358	Smart combo child order not supported.	
359	Combo order only supports reduce on fill without block(OCA).	

Code	TWS message	Additional notes
360	No whatif check support for smart combo order.	Pre-trade commissions and margin information is not available for this type of order.
361	Invalid trigger price.	
362	Invalid adjusted stop price.	
363	Invalid adjusted stop limit price.	
364	Invalid adjusted trailing amount.	
365	No scanner subscription found for ticker id:	Scanner market data subscription request with this ticker id has either been cancelled or is not found.
366	No historical data query found for ticker id:	Historical market data request with this ticker id has either been cancelled or is not found.
367	Volatility type if set must be 1 or 2 for VOL orders. Do not set it for other order types.	
368	Reference Price Type must be 1 or 2 for dynamic volatility management. Do not set it for non-VOL orders.	
369	Volatility orders are only valid for US options.	Make sure that you are placing an order for US OPT contract.
370	Dynamic Volatility orders must be SMART routed, or trade on a Price Improvement Exchange.	
371	VOL order requires positive floating point value for volatility. Do not set it for other order types.	
372	Cannot set dynamic VOL attribute on non-VOL order.	Make sure that your order type is 'VOL'.
373	Can only set stock range attribute on VOL or RELATIVE TO STOCK order.	
374	If both are set, the lower stock range attribute must be less than the upper stock range attribute.	
375	Stock range attributes cannot be negative.	
376	The order is not eligible for continuous update. The option must trade on a cheap-to-reroute exchange.	
377	Must specify valid delta hedge order aux. price.	
378	Delta hedge order type requires delta hedge aux. price to be specified.	Make sure your order has delta attribute.
379	Delta hedge order type requires that no delta hedge aux. price be specified.	Make sure you do not specify aux. delta hedge price.
380	This order type is not allowed for delta hedge orders.	Limit, Market or Relative orders are supported.
381	Your DDE.dll needs to be upgraded.	

Code	TWS message	Additional notes
382	The price specified violates the number of ticks constraint specified in the default order settings.	
383	The size specified violates the size constraint specified in the default order settings.	
384	Invalid DDE array request.	
385	Duplicate ticker ID for API scanner subscription.	Make sure you are using a unique ticker ID for your new scanner subscription.
386	Duplicate ticker ID for API historical data query.	Make sure you are using a unique ticker ID for your new historical market data query.
387	Unsupported order type for this exchange and security type.	You can review all order types and supported exchanges on the `Order Types and Algos` page.
388	Order size is smaller than the minimum requirement.	Check order size parameters for the specified contract from the TWS Contract Details.
389	Supplied routed order ID is not unique.	
390	Supplied routed order ID is invalid.	
391	The time or time-zone entered is invalid. The correct format is hh:mm:ss xxx	
392	Invalid order: contract expired.	You can not place an order for the expired contract.
393	Short sale slot may be specified for delta hedge orders only.	
394	Invalid Process Time: must be integer number of milliseconds between 100 and 2000. Found:	
395	Due to system problems, orders with OCA groups are currently not being accepted.	Check TWS bulletins for more information.
396	Due to system problems, application is currently accepting only Market and Limit orders for this contract.	Check TWS bulletins for more information.
397	Due to system problems, application is currently accepting only Market and Limit orders for this contract.	
398	$<>$ cannot be used as a condition trigger.	Please make sure that you specify a valid condition
399	Order message error	
400	Algo order error.	
401	Length restriction.	
402	Conditions are not allowed for this contract.	Condition order type does not support for this contract
403	Invalid stop price.	The Stop Price you specified for the order is invalid for the contract

Code	TWS message	Additional notes
404	Shares for this order are not immediately available for short sale. The order will be held while we attempt to locate the shares.	You order is held by the TWS because you are trying to sell a contract but you do not have any long position and the market does not have short sale available. You order will be transmitted once there is short sale available on the market
405	The child order quantity should be equivalent to the parent order size.	This error is deprecated.
406	The currency $< >$ is not allowed.	Please specify a valid currency
407	The symbol should contain valid non-unicode characters only.	Please check your contract Symbol
408	Invalid scale order increment.	
409	Invalid scale order. You must specify order component size.	ScaleInitLevelSize specified is invalid
410	Invalid subsequent component size for scale order.	ScaleSubsLevelSize specified is invalid
411	The "Outside Regular Trading Hours" flag is not valid for this order.	Trading outside of regular trading hours is not available for this security
412	The contract is not available for trading.	
413	What-if order should have the transmit flag set to true.	You need to set **IBApi.Order.↩ Transmit** to TRUE
414	Snapshot market data subscription is not applicable to generic ticks.	You must leave Generic Tick List to be empty when requesting snapshot market data
415	Wait until previous RFQ finishes and try again.	
416	RFQ is not applicable for the contract. Order ID:	
417	Invalid initial component size for scale order.	ScaleInitLevelSize specified is invalid
418	Invalid scale order profit offset.	ScaleProfitOffset specified is invalid
419	Missing initial component size for scale order.	You need to specify the ScaleInit↩ LevelSize
420	Invalid real-time query.	This error is deprecated.
421	Invalid route.	This error is deprecated.
422	The account and clearing attributes on this order may not be changed.	
423	Cross order RFQ has been expired. THI committed size is no longer available. Please open order dialog and verify liquidity allocation.	
424	FA Order requires allocation to be specified.	This error is deprecated.
425	FA Order requires per-account manual allocations because there is no common clearing instruction. Please use order dialog Adviser tab to enter the allocation.	This error is deprecated.
426	None of the accounts have enough shares.	You are not able to enter short position with Cash Account

Code	TWS message	Additional notes
427	Mutual Fund order requires monetary value to be specified.	This error is deprecated.
428	Mutual Fund Sell order requires shares to be specified.	This error is deprecated.
429	Delta neutral orders are only supported for combos (BAG security type).	
430	We are sorry, but fundamentals data for the security specified is not available.	
431	What to show field is missing or incorrect.	This error is deprecated.
432	Commission must not be negative.	This error is deprecated.
433	Invalid "Restore size after taking profit" for multiple account allocation scale order.	
434	The order size cannot be zero.	
435	You must specify an account.	The function you invoked only works on a single account
436	You must specify an allocation (either a single account, group, or profile).	When you try to place an order with a Financial Advisor account, you must specify the order to be routed to either a single account, a group, or a profile.
437	Order can have only one flag Outside RTH or Allow PreOpen.	This error is deprecated.
438	The application is now locked.	This error is deprecated.
439	Order processing failed. Algorithm definition not found.	Please double check your specification for **IBApi.Order.Algo↩ Strategy** and **IBApi.↩ Order.AlgoParams**
440	Order modify failed. Algorithm cannot be modified.	
441	Algo attributes validation failed:	Please double check your specification for **IBApi.Order.Algo↩ Strategy** and **IBApi.↩ Order.AlgoParams**
442	Specified algorithm is not allowed for this order.	
443	Order processing failed. Unknown algo attribute.	Specification for **IBApi.Order.↩ AlgoParams** is incorrect
444	Volatility Combo order is not yet acknowledged. Cannot submit changes at this time.	The order is not in a state that is able to be modified
445	The RFQ for this order is no longer valid.	
446	Missing scale order profit offset.	ScaleProfitOffset is not properly specified
447	Missing scale price adjustment amount or interval.	ScalePriceAdjustValue or Scale↩ PriceAdjustInterval is not specified properly
448	Invalid scale price adjustment interval.	ScalePriceAdjustInterval specified is invalid

Code	TWS message	Additional notes
449	Unexpected scale price adjustment amount or interval.	ScalePriceAdjustValue or Scale↩ PriceAdjustInterval specified is invalid
507	Bad Message Length (Java-only)	Indicates EOF exception was caught while reading from the socket. This can occur if there is an attempt to connect to TWS with a client ID that is already in use, or if TWS is locked, closes, or breaks the connection. It should be handled by the client application and used to indicate that the socket connection is not valid.
10000	Cross currency combo error.	
10001	Cross currency vol error.	
10002	Invalid non-guaranteed legs.	
10003	IBSX not allowed.	
10005	Read-only models.	
10006	Missing parent order.	The parent order ID specified cannot be found. In some cases this can occur with bracket orders if the child order is placed immediately after the parent order; a brief pause of 50 ms or less will be necessary before the child order is transmitted to TWS/IBG.
10007	Invalid hedge type.	
10008	Invalid beta value.	
10009	Invalid hedge ratio.	
10010	Invalid delta hedge order.	
10011	Currency is not supported for Smart combo.	
10012	Invalid allocation percentage	FaPercentage specified is not valid
10013	Smart routing API error (Smart routing opt-out required).	
10014	PctChange limits.	This error is deprecated
10015	Trading is not allowed in the API.	
10016	Contract is not visible.	This error is deprecated
10017	Contracts are not visible.	This error is deprecated
10018	Orders use EV warning.	
10019	Trades use EV warning.	
10020	Display size should be smaller than order size./td>	The display size should be smaller than the total quantity
10021	Invalid leg2 to Mkt Offset API.	This error is deprecated
10022	Invalid Leg Prio API.	This error is deprecated
10023	Invalid combo display size API.	This error is deprecated
10024	Invalid don't start next legin API.	This error is deprecated
10025	Invalid leg2 to Mkt time1 API.	This error is deprecated
10026	Invalid leg2 to Mkt time2 API.	This error is deprecated
10027	Invalid combo routing tag API.	This error is deprecated

Code	TWS message	Additional notes
10090	Part of requested market data is not subscribed.	Indicates that some tick types requested require additional market data subscriptions not held in the account. This commonly occurs for instance if a user has options subscriptions but not the underlying stock so the system cannot calculate the real time greek values (other default ticks will be returned). Or alternatively, if generic tick types are specified in a market data request without the associated subscriptions.

Chapter 17

Market Scanners

http://interactivebrokers.github.io/tws-api/market_scanners.html

Some scans in the TWS `Advanced Market Scanner` can be accessed via the TWS API through the **IBApi.EClient.reqScannerSubscription**.

- ```
  client.reqScannerSubscription(7001, ScannerSubscriptionSamples.HighOptVolumePCRatioUSIndexes(), null);
  ```

- ```
  client.reqScannerSubscription(7001, ScannerSubscriptionSamples.HighOptVolumePCRatioUSIndexes(), null);
  ```

- ```
  client.reqScannerSubscription(7001, ScannerSubscriptionSamples.HighOptVolumePCRatioUSIndexes(),
  Nothing)
  ```

- ```
  m_pClient->reqScannerSubscription(7001, ScannerSubscriptionSamples::HotUSStkByVolume(),
  TagValueListSPtr());
  ```

- ```
  self.reqScannerSubscription(7001,
                                  ScannerSubscriptionSamples.HighOptVolumePCRatioUSIndexes(), [])
  ```

The available API filters and and parameters are defined in the **IBApi.ScannerSubscription** object, including [AbovePrice, BelowPrice, AboveVolume, ...] . Some **Market Scanner Examples** are listed at the bottom of this page.

Scans are limited to a maximum result of 50 results per scan code, and only 10 API scans can be active at a time.

Results are delivered via **IBApi.EWrapper.scannerData** and the **IBApi.EWrapper.scannerDataEnd** marker will indicate when all results have been delivered. The returned results to scannerData simply consist of a list of contracts. There are no market data fields (bid, ask, last, volume, ...) returned from the scanner, and so if these are desired they have to be requested separately with the reqMktData function. Since the scanner results do not include any market data fields, it is not necessary to have market data subscriptions to use the API scanner. However to use filters, market data subscriptions **are** generally required.

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
    public virtual void scannerData(int reqId, int rank, ContractDetails contractDetails, string
  distance, string benchmark, string projection, string legsStr)
    {
        Console.WriteLine("ScannerData. "+reqId+" - Rank: "+rank+", Symbol: "+contractDetails.Contract.
  Symbol+", SecType: "+contractDetails.Contract.SecType+", Currency: "+contractDetails.Contract.Currency
            +", Distance: "+distance+", Benchmark: "+benchmark+", Projection: "+projection+", Legs
   String: "+legsStr);
    }
  ```

  ...

```
        public virtual void scannerDataEnd(int reqId)
        {
            Console.WriteLine("ScannerDataEnd. "+reqId);
        }
```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void scannerData(int reqId, int rank,
              ContractDetails contractDetails, String distance, String benchmark,
              String projection, String legsStr) {
          System.out.println("ScannerData. "+reqId+" - Rank: "+rank+", Symbol: "+contractDetails.contract().
        symbol()+", SecType: "+contractDetails.contract().secType()+", Currency: "+contractDetails.contract().
        currency()
                    +", Distance: "+distance+", Benchmark: "+benchmark+", Projection: "+projection+", Legs
         String: "+legsStr);
      }
  ```

  ...

  ```
      @Override
      public void scannerDataEnd(int reqId) {
          System.out.println("ScannerDataEnd. "+reqId);
      }
  ```

- ```
      Public Class EWrapperImpl
          Implements EWrapper
  ```

  ...

  ```
          Public Sub scannerData(reqId As Integer, rank As Integer, contractDetails As IBApi.ContractDetails,
         distance As String, benchmark As String, projection As String, legsStr As String) Implements
         IBApi.EWrapper.scannerData
              Console.WriteLine("ScannerData. " & reqId & " - Rank: " & rank & ", Symbol: " &
         contractDetails.Contract.Symbol & ", SecType: " &
                               contractDetails.Contract.SecType & ", Currency: " &
         contractDetails.Contract.Currency & ", Distance: " & distance &
                               ", Benchmark: " & benchmark & ", Projection: " & projection & ", Legs String: " &
         legsStr)
          End Sub
  ```

  ...

  ```
          Public Sub scannerDataEnd(reqId As Integer) Implements IBApi.EWrapper.scannerDataEnd
              Console.WriteLine("ScannerDataEnd. " & reqId & "\n")
          End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::scannerData(int reqId, int rank, const ContractDetails& contractDetails,
                                  const std::string& distance, const std::string& benchmark, const
        std::string& projection,
                                  const std::string& legsStr) {
      printf( "ScannerData. %d - Rank: %d, Symbol: %s, SecType: %s, Currency: %s, Distance: %s, Benchmark:
         %s, Projection: %s, Legs String: %s\n", reqId, rank, contractDetails.contract.symbol.c_str(), contractDetails.
        contract.secType.c_str(), contractDetails.contract.currency.c_str(), distance.c_str(), benchmark.c_str(),
        projection.c_str(), legsStr.c_str());
  }
  ```

  ...

  ```
  void TestCppClient::scannerDataEnd(int reqId) {
      printf( "ScannerDataEnd. %d\n", reqId);
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

```
    def scannerData(self, reqId: int, rank: int, contractDetails: ContractDetails,
                    distance: str, benchmark: str, projection: str, legsStr: str):
        super().scannerData(reqId, rank, contractDetails, distance, benchmark,
                            projection, legsStr)
        print("ScannerData. ", reqId, "Rank:", rank, "Symbol:", contractDetails.contract.symbol,
              "SecType:", contractDetails.contract.secType,
              "Currency:", contractDetails.contract.currency,
              "Distance:", distance, "Benchmark:", benchmark,
              "Projection:", projection, "Legs String:", legsStr)
```

...

```
    def scannerDataEnd(self, reqId: int):
        super().scannerDataEnd(reqId)
        print("ScannerDataEnd. ", reqId)
```

Since the **IBApi.EClient.reqScannerSubscription** request keeps a subscription open you will keep receiv-ing periodic updates until the request is cancelled via **IBApi.EClient.cancelScannerSubscription** :

- ```
            client.cancelScannerSubscription(7001);
  ```
- ```
        client.cancelScannerSubscription(7001);
  ```
- ```
        client.cancelScannerSubscription(7001)
  ```
- ```
    m_pClient->cancelScannerSubscription(7001);
  ```
- ```
        self.cancelScannerSubscription(7001)
  ```

18.1 Market Scanner Examples

The available API scans are defined by the fields in the ScannerSubscription object, i.e. *abovePrice, aboveVolume, belowPrice*, etc. Some market scanner examples are below.

- ```
            //Hot US stocks by volume
            ScannerSubscription scanSub = new ScannerSubscription();
            scanSub.Instrument = "STK";
            scanSub.LocationCode = "STK.US.MAJOR";
            scanSub.ScanCode = "HOT_BY_VOLUME";
  ```

...

```
            //Top % gainers at IBIS
            ScannerSubscription scanSub = new ScannerSubscription();
            scanSub.Instrument = "STOCK.EU";
            scanSub.LocationCode = "STK.EU.IBIS";
            scanSub.ScanCode = "TOP_PERC_GAIN";
```

...

```
            //Most active futures at SOFFEX
            ScannerSubscription scanSub = new ScannerSubscription();
            scanSub.Instrument = "FUT.EU";
            scanSub.LocationCode = "FUT.EU.SOFFEX";
            scanSub.ScanCode = "MOST_ACTIVE";
```

...

```
            //High option volume P/C ratio US indexes
            ScannerSubscription scanSub = new ScannerSubscription();
            scanSub.Instrument = "IND.US";
            scanSub.LocationCode = "IND.US";
            scanSub.ScanCode = "HIGH_OPT_VOLUME_PUT_CALL_RATIO";
```

- ```
  //Hot US stocks by volume
  ScannerSubscription scanSub = new ScannerSubscription();
  scanSub.instrument("STK");
  scanSub.locationCode("STK.US.MAJOR");
  scanSub.scanCode("HOT_BY_VOLUME");
  ```

...

```
//Top % gainers at IBIS
ScannerSubscription scanSub = new ScannerSubscription();
scanSub.instrument("STOCK.EU");
scanSub.locationCode("STK.EU.IBIS");
scanSub.scanCode("TOP_PERC_GAIN");
```

...

```
//Most active futures at SOFFEX
ScannerSubscription scanSub = new ScannerSubscription();
scanSub.instrument("FUT.EU");
scanSub.locationCode("FUT.EU.SOFFEX");
scanSub.scanCode("MOST_ACTIVE");
```

...

```
//High option volume P/C ratio US indexes
ScannerSubscription scanSub = new ScannerSubscription();
scanSub.instrument("IND.US");
scanSub.locationCode("IND.US");
scanSub.scanCode("HIGH_OPT_VOLUME_PUT_CALL_RATIO");
```

- ```
  'Hot US stocks by volume
  Dim scanSub As ScannerSubscription = New ScannerSubscription()
  scanSub.Instrument = "STK"
  scanSub.LocationCode = "STK.US.MAJOR"
  scanSub.ScanCode = "HOT_BY_VOLUME"
  ```

...

```
'Top % gainers at IBIS
Dim scanSub As ScannerSubscription = New ScannerSubscription()
scanSub.Instrument = "STOCK.EU"
scanSub.LocationCode = "STK.EU.IBIS"
scanSub.ScanCode = "TOP_PERC_GAIN"
```

...

```
'Most active futures at SOFFEX
Dim scanSub As ScannerSubscription = New ScannerSubscription()
scanSub.Instrument = "FUT.EU"
scanSub.LocationCode = "FUT.EU.SOFFEX"
scanSub.ScanCode = "MOST_ACTIVE"
```

...

```
'High option volume P/C ratio US indexes
Dim scanSub As ScannerSubscription = New ScannerSubscription()
scanSub.Instrument = "IND.US"
scanSub.LocationCode = "IND.US"
scanSub.ScanCode = "HIGH_OPT_VOLUME_PUT_CALL_RATIO"
```

- ```
  //Hot US stocks by volume
  ScannerSubscription scanSub;
  scanSub.instrument = "STK";
  scanSub.locationCode = "STK.US.MAJOR";
  scanSub.scanCode = "HOT_BY_VOLUME";
  ```

...

```
//Top % gainers at IBIS
ScannerSubscription scanSub;
scanSub.instrument = "STOCK.EU";
scanSub.locationCode = "STK.EU.IBIS";
scanSub.scanCode = "TOP_PERC_GAIN";
```

...
```

```
        //Most active futures at SOFFEX
        ScannerSubscription scanSub;
        scanSub.instrument = "FUT.EU";
        scanSub.locationCode = "FUT.EU.SOFFEX";
        scanSub.scanCode = "MOST_ACTIVE";
```

...

```
        //High option volume P/C ratio US indexes
        ScannerSubscription scanSub;
        scanSub.instrument = "IND.US";
        scanSub.locationCode = "IND.US";
        scanSub.scanCode = "HIGH_OPT_VOLUME_PUT_CALL_RATIO";
```

-
```
        #Hot US stocks by volume
        scanSub = ScannerSubscription()
        scanSub.instrument = "STK"
        scanSub.locationCode = "STK.US.MAJOR"
        scanSub.scanCode = "HOT_BY_VOLUME"
```

...

```
        # Top % gainers at IBIS
        scanSub = ScannerSubscription()
        scanSub.instrument = "STOCK.EU"
        scanSub.locationCode = "STK.EU.IBIS"
        scanSub.scanCode = "TOP_PERC_GAIN"
```

...

```
        # Most active futures at SOFFEX
        scanSub = ScannerSubscription()
        scanSub.instrument = "FUT.EU"
        scanSub.locationCode = "FUT.EU.SOFFEX"
        scanSub.scanCode = "MOST_ACTIVE"
```

...

```
        # High option volume P/C ratio US indexes
        scanSub = ScannerSubscription()
        scanSub.instrument = "IND.US"
        scanSub.locationCode = "IND.US"
        scanSub.scanCode = "HIGH_OPT_VOLUME_PUT_CALL_RATIO"
```

See also **Scanner Parameters**

18.2 Scanner Parameters

A list of *TWS* scanner parameters can be obtained via **IBApi.EClient.reqScannerParameters** (p. **??**). Not all of the returned parameters to scannerParameters are available from the API scanners.

-
```
            client.reqScannerParameters();
```

-
```
        client.reqScannerParameters();
```

-
```
        client.reqScannerParameters()
```

-
```
    m_pClient->reqScannerParameters();
```

-
```
        self.reqScannerParameters()
```

A string containing all available XML-formatted parameters will then be returned via **IBApi.EWrapper.scanner↩ Parameters**

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
      public virtual void scannerParameters(string xml)
      {
          Console.WriteLine("ScannerParameters. "+xml+"\n");
      }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
      @Override
      public void scannerParameters(String xml) {
          System.out.println("ScannerParameters. "+xml+"\n");
      }
  ```

- ```
  Public Class EWrapperImpl
      Implements EWrapper
  ```

  ...

  ```
      Public Sub scannerParameters(xml As String) Implements IBApi.EWrapper.scannerParameters
          Console.WriteLine("ScannerParameters. " & xml & "\n")
      End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::scannerParameters(const std::string& xml) {
      printf( "ScannerParameters. %s\n", xml.c_str());
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def scannerParameters(self, xml: str):
          super().scannerParameters(xml)
          open('log/scanner.xml', 'w').write(xml)
  ```

Important: remember the TWS API is just an interface to the TWS. If you are having problems defining a scanner, always make sure you can create a similar scanner using the TWS' `Advanced Market Scanner`.
  ```

Chapter 18

News

http://interactivebrokers.github.io/tws-api/news.html

API news requires news subscriptions that are specific to the API; most news services in TWS are not also available in the API. Beginning in *TWS v966*, three API news services are enabled in accounts by default and available from the API. They are:

- Briefing.com General Market Columns (BRFG)
- Briefing.com Analyst Actions (BRFUPDN)
- Dow Jones Newsletters (DJNL)

There are also four additional news services available with all TWS versions which require **API-specific subscriptions** to first be made in Account Management. They have different data fees than the subscription for the same news in TWS-only. As with all subscriptions, they only apply to the specific TWS username under which they were made:

- Briefing Trader (BT)
- Benzinga Pro (BZ)
- Fly on the Wall (FLY)
- Midnight Trader (MT) ∗not currently available from the API

The API functions which handle news are able to query available news provides, subscribe to news in real time to receive headlines as they are released, request specific news articles, and return a historical list of news stories that are cached in the system.

19.1 Checking Subscribed News Sources

Adding or removing API news subscriptions from an account is accomplished through Account Management. From the API, currently subscribed news sources can be retrieved using the function **IBApi::EClient::reqNewsProviders** (p. **??**). Note this function is only available starting in API v973.02.

- ```
  client.reqNewsProviders();
  ```
- ```
  client.reqNewsProviders();
  ```
- ```
  client.reqNewsProviders()
  ```
  ```

- ```
  m_pClient->reqNewsProviders();
  ```
- ```
  self.reqNewsProviders()
  ```

A list of available subscribed news sources is returned to the function **IBApi::EWrapper::newsProviders**

- ```
  public void newsProviders(NewsProvider[] newsProviders)
  {
      Console.WriteLine("News Providers:");

      foreach (var newsProvider in newsProviders)
      {
          Console.WriteLine("News provider: providerCode - {0}, providerName - {1}",
              newsProvider.ProviderCode, newsProvider.ProviderName);
      }
  }
  ```
- ```
  @Override
  public void newsProviders(NewsProvider[] newsProviders) {
      for (NewsProvider np : newsProviders) {
          System.out.print("News Provider. ProviderCode: " + np.providerCode() + ", ProviderName: " + np.
    providerName() + "\n");
      }

      System.out.println();
  }
  ```
- ```
  Public Sub newsProviders(newsProviders As NewsProvider()) Implements EWrapper.newsProviders
      Console.WriteLine("News Providers")

      For Each newsProvider In newsProviders
          Console.WriteLine("News Provider: providerCode - " & newsProvider.ProviderCode & ",
   providerName - " & newsProvider.ProviderName)
      Next
  End Sub
  ```
- ```
  void TestCppClient::newsProviders(const std::vector<NewsProvider> &newsProviders) {
      printf("News providers (%lu):\n", newsProviders.size());

      for (unsigned int i = 0; i < newsProviders.size(); i++) {
          printf("News provider [%d] - providerCode: %s providerName: %s\n", i, newsProviders[i].providerCode
    .c_str(), newsProviders[i].providerName.c_str());
      }
  }
  ```
- ```
  def newsProviders(self, newsProviders: ListOfNewsProviders):
      print("newsProviders: ")
      for provider in newsProviders:
          print(provider)
  ```

19.2 Contract Specific News

There are two ways to subscribe to real time news headlines through the API. You can request **Contract** specific News or **BroadTape** News. See also **BroadTape News List**

Important: in order to obtain news feeds via the TWS API you need to acquire the relevant API-specific subscriptions via your Account Management.

When invoking **IBApi.EClient.reqMktData** (p. **??**), for a specific **IBApi.Contract** you will follow the same format convention as any other **Basic Contracts** (p. **??**). The News Source is identified by the **genericTickList**.

Note: The error message "invalid tick type" will be returned if the username has not added the appropriate API news subscription.

- ```
  // Without the API news subscription this will generate an "invalid tick type" error
  client.reqMktData(1005, ContractSamples.USStock(), "mdoff,292:BZ", false, false, null);
  client.reqMktData(1006, ContractSamples.USStock(), "mdoff,292:BT", false, false, null);
  client.reqMktData(1007, ContractSamples.USStock(), "mdoff,292:FLY", false, false, null);
  client.reqMktData(1008, ContractSamples.USStock(), "mdoff,292:MT", false, false, null);
  ```

```
        // Without the API news subscription this will generate an "invalid tick type" error
        client.reqMktData(1005, ContractSamples.USStock(), "mdoff,292:BZ", false, false, null);
        client.reqMktData(1006, ContractSamples.USStock(), "mdoff,292:BT", false, false, null);
        client.reqMktData(1007, ContractSamples.USStock(), "mdoff,292:FLY", false, false, null);
        client.reqMktData(1008, ContractSamples.USStock(), "mdoff,292:MT", false, false, null);
```

```
        ' Without the API news subscription this will generate an "invalid tick type" error
        client.reqMktData(1005, ContractSamples.USStock(), "mdoff,292:BZ", False, False, Nothing)
        client.reqMktData(1006, ContractSamples.USStock(), "mdoff,292:BT", False, False, Nothing)
        client.reqMktData(1007, ContractSamples.USStock(), "mdoff,292:FLY", False, False, Nothing)
        client.reqMktData(1008, ContractSamples.USStock(), "mdoff,292:MT", False, False, Nothing)
```

```
    // Without the API news subscription this will generate an "invalid tick type" error
    m_pClient->reqMktData(1005, ContractSamples::USStock(), "mdoff,292:BZ", false, false, TagValueListSPtr(
      ));
    m_pClient->reqMktData(1006, ContractSamples::USStock(), "mdoff,292:BT", false, false, TagValueListSPtr(
      ));
    m_pClient->reqMktData(1007, ContractSamples::USStock(), "mdoff,292:FLY", false, false, TagValueListSPtr
      ());
    m_pClient->reqMktData(1008, ContractSamples::USStock(), "mdoff,292:MT", false, false, TagValueListSPtr(
      ));
```

```
        # Without the API news subscription this will generate an "invalid tick type" error
        self.reqMktData(1005, ContractSamples.USStock(), "mdoff,292:BZ", False, False, [])
        self.reqMktData(1006, ContractSamples.USStock(), "mdoff,292:BT", False, False, [])
        self.reqMktData(1007, ContractSamples.USStock(), "mdoff,292:FLY", False, False, [])
        self.reqMktData(1008, ContractSamples.USStock(), "mdoff,292:MT", False, False, [])
```

BroadTape News

For **BroadTape** News you specify the **IBApi.Contract** for the specific news source. This is uniquely identified by the symbol and exchange. The symbol of an instrument can easily be obtained via the **IBApi.EClientSocket.↩ reqContractDetails** request.

Examples of news contracts

Briefing Trader

```
            Contract contract = new Contract();
            contract.Symbol = "BT:BT_ALL"; //BroadTape All News
            contract.SecType = "NEWS";
            contract.Exchange = "BT"; //Briefing Trader
```

```
        Contract contract = new Contract();
        contract.symbol("BT:BT_ALL"); //BroadTape All News
        contract.secType("NEWS");
        contract.exchange("BT"); //Briefing Trader
```

```
            Dim contract As Contract = New Contract()
            contract.Symbol = "BT:BT_ALL" 'BroadTape All News
            contract.SecType = "NEWS"
            contract.Exchange = "BT" 'Briefing Trader
```

```
    Contract contract;
    contract.symbol = "BT:BT_ALL"; //BroadTape All News
    contract.secType = "NEWS";
    contract.exchange = "BT"; //Briefing Trader
```

```
        contract = Contract()
        contract.symbol  = "BT:BT_ALL" #BroadTape All News
        contract.secType = "NEWS"
        contract.exchange = "BT" #Briefing Trader
```

Benzinga Pro

- ```
            Contract contract = new Contract();
            contract.Symbol = "BZ:BZ_ALL"; //BroadTape All News
            contract.SecType = "NEWS";
            contract.Exchange = "BZ"; //Benzinga Pro
  ```

- ```
        Contract contract = new Contract();
        contract.symbol("BZ:BZ_ALL"); //BroadTape All News
        contract.secType("NEWS");
        contract.exchange("BZ"); //Benzinga Pro
  ```

- ```
            Dim contract As Contract = New Contract()
            contract.Symbol = "BZ:BZ_ALL" 'BroadTape All News
            contract.SecType = "NEWS"
            contract.Exchange = "BZ" 'Benzinga Pro
  ```

- ```
    Contract contract;
    contract.symbol = "BZ:BZ_ALL"; //BroadTape All News
    contract.secType = "NEWS";
    contract.exchange = "BZ"; //Benzinga Pro
  ```

- ```
        contract = Contract()
        contract.symbol = "BZ:BZ_ALL" #BroadTape All News
        contract.secType = "NEWS"
        contract.exchange = "BZ" #Benzinga Pro
  ```

Fly on the Wall

- ```
            Contract contract = new Contract();
            contract.Symbol = "FLY:FLY_ALL"; //BroadTape All News
            contract.SecType = "NEWS";
            contract.Exchange = "FLY"; //Fly on the Wall
  ```

- ```
        Contract contract = new Contract();
        contract.symbol("FLY:FLY_ALL"); //BroadTape All News
        contract.secType("NEWS");
        contract.exchange("FLY"); //Fly on the Wall
  ```

- ```
            Dim contract As Contract = New Contract()
            contract.Symbol = "FLY:FLY_ALL" 'BroadTape All News
            contract.SecType = "NEWS"
            contract.Exchange = "FLY" 'Fly On the Wall
  ```

- ```
    Contract contract;
    contract.symbol = "FLY:FLY_ALL"; //BroadTape All News
    contract.secType = "NEWS";
    contract.exchange = "FLY"; //Fly on the Wall
  ```

- ```
        contract = Contract()
        contract.symbol  = "FLY:FLY_ALL" #BroadTape All News
        contract.secType = "NEWS"
        contract.exchange = "FLY" #Fly on the Wall
  ```

Midnight Trader

- ```
            Contract contract = new Contract();
            contract.Symbol = "MT:MT_ALL"; //BroadTape All News
            contract.SecType = "NEWS";
            contract.Exchange = "MT"; //Midnight Trader
  ```

- ```
        Contract contract = new Contract();
        contract.symbol("MT:MT_ALL"); //BroadTape All News
        contract.secType("NEWS");
        contract.exchange("MT"); //Midnight Trader
  ```

- ```
            Dim contract As Contract = New Contract()
            contract.Symbol = "MT:MT_ALL" 'BroadTape All News
            contract.SecType = "NEWS"
            contract.Exchange = "MT" 'Midnight Trader
  ```

- ```
    Contract contract;
    contract.symbol = "MT:MT_ALL"; //BroadTape All News
    contract.secType = "NEWS";
    contract.exchange = "MT"; //Midnight Trader
  ```

- ```
        contract = Contract()
        contract.symbol = "MT:MT_ALL" #BroadTape All News
        contract.secType = "NEWS"
        contract.exchange = "MT" #Midnight Trader
  ```
  ```

Then you can invoke **IBApi.EClient.reqMktData** for the broad tape news feed.

```
            client.reqMktData(1009, ContractSamples.BTbroadtapeNewsFeed(), "mdoff,292", false, false, null)
        ;
            client.reqMktData(1010, ContractSamples.BZbroadtapeNewsFeed(), "mdoff,292", false, false, null)
        ;
            client.reqMktData(1011, ContractSamples.FLYbroadtapeNewsFeed(), "mdoff,292", false, false, null
        );
            client.reqMktData(1012, ContractSamples.MTbroadtapeNewsFeed(), "mdoff,292", false, false, null)
        ;
```

```
        client.reqMktData(1009, ContractSamples.BTbroadtapeNewsFeed(), "mdoff,292", false, false, null);
        client.reqMktData(1010, ContractSamples.BZbroadtapeNewsFeed(), "mdoff,292", false, false, null);
        client.reqMktData(1011, ContractSamples.FLYbroadtapeNewsFeed(), "mdoff,292", false, false, null);
        client.reqMktData(1012, ContractSamples.MTbroadtapeNewsFeed(), "mdoff,292", false, false, null);
```

```
        client.reqMktData(1009, ContractSamples.BTbroadtapeNewsFeed(), "mdoff,292", False, False, Nothing)
        client.reqMktData(1010, ContractSamples.BZbroadtapeNewsFeed(), "mdoff,292", False, False, Nothing)
        client.reqMktData(1011, ContractSamples.FLYbroadtapeNewsFeed(), "mdoff,292", False, False, Nothing)
        client.reqMktData(1012, ContractSamples.MTbroadtapeNewsFeed(), "mdoff,292", False, False, Nothing)
```

```
    m_pClient->reqMktData(1009, ContractSamples::BTbroadtapeNewsFeed(), "mdoff,292", false, false,
      TagValueListSPtr());
    m_pClient->reqMktData(1010, ContractSamples::BZbroadtapeNewsFeed(), "mdoff,292", false, false,
      TagValueListSPtr());
    m_pClient->reqMktData(1011, ContractSamples::FLYbroadtapeNewsFeed(), "mdoff,292", false, false,
      TagValueListSPtr());
    m_pClient->reqMktData(1012, ContractSamples::MTbroadtapeNewsFeed(), "mdoff,292", false, false,
      TagValueListSPtr());
```

```
        self.reqMktData(1009, ContractSamples.BTbroadtapeNewsFeed(),
                        "mdoff,292", False, False, [])
        self.reqMktData(1010, ContractSamples.BZbroadtapeNewsFeed(),
                        "mdoff,292", False, False, [])
        self.reqMktData(1011, ContractSamples.FLYbroadtapeNewsFeed(),
                        "mdoff,292", False, False, [])
        self.reqMktData(1012, ContractSamples.MTbroadtapeNewsFeed(),
                        "mdoff,292", False, False, [])
```

Important: For Briefing Trader live head lines via the API is only offered on a case-by-case basis directly from Briefing.com offers Briefing Trader subscribers access to the subscription live head lines via the A↩ PI. For more information and to submit an API entitlement application, please contact Briefing.com directly at dbeasley@briefing.com.

The resulting news headlines are returned to the function **IBApi::EWrapper::tickNews** with the news head-line, timestamp, and article ID. The article ID can be used to retrieve the body of the news article using the function **IBApi::EClient::reqNewsArticle** below.

```
        public void tickNews(int tickerId, long timeStamp, string providerCode, string articleId, string
      headline, string extraData)
        {
            Console.WriteLine("Tick News. Ticker Id: {0}, Time Stamp: {1}, Provider Code: {2}, Article Id:
       {3}, headline: {4}, extraData: {5}", tickerId, timeStamp, providerCode, articleId, headline, extraData);
        }
```

```
    @Override
    public void tickNews(int tickerId, long timeStamp, String providerCode, String articleId, String
      headline, String extraData) {
        System.out.println("Tick News. TickerId: " + tickerId + ", TimeStamp: " + timeStamp + ",
       ProviderCode: " + providerCode + ", ArticleId: " + articleId + ", Headline: " + headline + ", ExtraData: " + extr
       + "\n");
    }
```

```
        Public Sub tickNews(tickerId As Integer, timeStamp As Long, providerCode As String, articleId As
       String, headline As String, extraData As String) Implements IBApi.EWrapper.tickNews
            Console.WriteLine("Tick News. Ticker Id: " & tickerId & ", Time Stamp: " & timeStamp & ",
       Provider Code: " & providerCode & ", Article Id: " & articleId & ", Headline: " & headline & ", Extra Data: " &
       extraData)
        End Sub
```

```
void TestCppClient::tickNews(int tickerId, time_t timeStamp, const std::string& providerCode, const
      std::string& articleId, const std::string& headline, const std::string& extraData) {
    printf("News Tick. TickerId: %d, TimeStamp: %s, ProviderCode: %s, ArticleId: %s, Headline: %s,
       ExtraData: %s\n", tickerId, ctime(&(timeStamp /= 1000)), providerCode.c_str(), articleId.c_str(), headline.c_str
      extraData.c_str());
}
```

```
    def tickNews(self, tickerId: int, timeStamp: int, providerCode: str,
                 articleId: str, headline: str, extraData: str):
        print("tickNews: ", tickerId, ", timeStamp: ", timeStamp,
              ", providerCode: ", providerCode, ", articleId: ", articleId,
              ", headline: ", headline, "extraData: ", extraData)
```

19.3 Historical News Headlines

With the appropriate API news subscription, historical news headlines can be requested from the API using the function **IBApi::EClient::reqHistoricalNews** starting in API v973.02. The resulting headlines are returned to **IBApi::EWrapper::historicalNews** (p. **??**).

- ```
            client.reqHistoricalNews(12003, 8314, "BZ+FLY", "", "", 10, null);
  ```
- ```
        client.reqHistoricalNews(10003, 8314, "BZ+FLY", "", "", 10, null);
  ```
- ```
        client.reqHistoricalNews(10003, 8314, "BZ+FLY", "", "", 10, Nothing)
  ```
- ```
    TagValueList* list = new TagValueList();
    list->push_back((TagValueSPtr)new TagValue("manual", "1"));
    m_pClient->reqHistoricalNews(12001, 8314, "BZ+FLY", "", "", 5, TagValueListSPtr(list));
  ```
- ```
        self.reqHistoricalNews(215, 8314, "BZ+FLY", "", "", 10, [])
  ```

19.4 Requesting News Articles

After requesting news headlines using one of the above functions, the body of a news article can be requested with the article ID returned by invoking the function **IBApi::EClient::reqNewsArticle** starting in API v973.02. The body of the news article is returned to the function **IBApi::EWrapper::newsArticle** (p. **??**).

- ```
            client.reqNewsArticle(12002, "BZ", "BZ$04507322", null);
  ```
- ```
        client.reqNewsArticle(10002, "BZ", "BZ$04507322", null);
  ```
- ```
        client.reqNewsArticle(10002, "BZ", "BZ$04507322", Nothing)
  ```
- ```
    TagValueList* list = new TagValueList();
    // list->push_back((TagValueSPtr)new TagValue("manual", "1"));
    m_pClient->reqNewsArticle(12001, "MST", "MST$06f53098", TagValueListSPtr(list));
  ```
- ```
        self.reqNewsArticle(214,"BZ", "BZ$04507322", [])
  ```

Chapter 19

IB Bulletins

http://interactivebrokers.github.io/tws-api/ib_bulletins.html

From time to time, IB sends out important `News Bulletins`, which can be accessed via the TWS API through the **IBApi.EClient.reqNewsBulletins** (p. **??**).

- ```
            client.reqNewsBulletins(true);
  ```
- ```
        client.reqNewsBulletins(true);
  ```
- ```
        client.reqNewsBulletins(True)
  ```
- ```
    m_pClient->reqNewsBulletins(true);
  ```
- ```
        self.reqNewsBulletins(True)
  ```

Bulletins are delivered via **IBApi.EWrapper.updateNewsBulletin** whenever there is a new bulletin:

- ```
    public class EWrapperImpl : EWrapper
    {
  ```

  ...

  ```
        public virtual void updateNewsBulletin(int msgId, int msgType, String message, String origExchange)
        {
            Console.WriteLine("News Bulletins. "+msgId+" - Type: "+msgType+", Message: "+message+",
       Exchange of Origin: "+origExchange+"\n");
        }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...

  ```
    @Override
    public void updateNewsBulletin(int msgId, int msgType, String message,
            String origExchange) {
        System.out.println("News Bulletins. "+msgId+" - Type: "+msgType+", Message: "+message+", Exchange
       of Origin: "+origExchange+"\n");
    }
  ```

- ```
    Public Class EWrapperImpl
        Implements EWrapper
  ```

  ...

  ```
        Public Sub updateNewsBulletin(msgId As Integer, msgType As Integer, message As String, origExchange
       As String) Implements IBApi.EWrapper.updateNewsBulletin
            Console.WriteLine("News Bulletins. " & msgId & " - Type: " & msgType & ", Message: " & message
       & ", Exchange of Origin: " & origExchange)
        End Sub
  ```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...
  ```

```
void TestCppClient::updateNewsBulletin(int msgId, int msgType, const std::string& newsMessage, const
      std::string& originExch) {
    printf( "News Bulletins. %d - Type: %d, Message: %s, Exchange of Origin: %s\n", msgId, msgType,
      newsMessage.c_str(), originExch.c_str());
}
```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def updateNewsBulletin(self, msgId: int, msgType: int, newsMessage: str,
                             originExch: str):
          super().updateNewsBulletin(msgId, msgType, newsMessage, originExch)
          print("News Bulletins. ", msgId, " Type: ", msgType, "Message:", newsMessage,
                "Exchange of Origin: ", originExch)
  ```

In order to stop receiving bulletins you need to cancel the subscription:

- ```
              client.cancelNewsBulletin();
  ```
- ```
          client.cancelNewsBulletins();
  ```
- ```
          client.cancelNewsBulletin()
  ```
- ```
      m_pClient->cancelNewsBulletins();
  ```
- ```
          self.cancelNewsBulletins()
  ```

Chapter 20

Display Groups

http://interactivebrokers.github.io/tws-api/display_groups.html

Display Groups function allows API clients to integrate with `TWS Color Grouping Windows`.

TWS Color Grouping Windows are identified by a colored chain in TWS and by an integer number via the API. Currently that number ranges from 1 to 7 and are mapped to specific colors, as indicated in TWS.

21.1 Query Display Groups

The **IBApi.EClient.queryDisplayGroups** method is used to request all available Display Groups in TWS.

- ```
  client.queryDisplayGroups(9001);
  ```
- ```
  client.queryDisplayGroups(9001);
  ```
- ```
  client.queryDisplayGroups(9001)
  ```
- ```
  m_pClient->queryDisplayGroups(9001);
  ```
- ```
  self.queryDisplayGroups(19001)
  ```

The **IBApi.EWrapper.displayGroupList** is a one-time response to **IBApi.EClient.queryDisplayGroups** (p. **??**).

It returns a list of integers representing visible Group ID separated by the "|" character, and sorted by most used group first. This list will not change during TWS session. In other words, user cannot add a new group, but only the sorting of the group numbers can change.

Example: "4|1|2|5|3|6|7"

- ```
  public class EWrapperImpl : EWrapper
  {
  ```

  ...

  ```
  public virtual void displayGroupList(int reqId, string groups)
  {
      Console.WriteLine("DisplayGroupList. Request: " + reqId + ", Groups" + groups);
  }
  ```

- ```
  public class EWrapperImpl implements EWrapper {
  ```

  ...
  ```

```
    @Override
    public void displayGroupList(int reqId, String groups) {
        System.out.println("Display Group List. ReqId: "+reqId+", Groups: "+groups+"\n");
    }
```

```
        Public Sub displayGroupList(reqId As Integer, groups As String) Implements
    IBApi.EWrapper.displayGroupList
        Console.WriteLine("DisplayGroupList - ReqId [" & reqId & "] Groups [" & groups & "]")
    End Sub
```

```
class TestCppClient : public EWrapper
{
```

...

```
void TestCppClient::displayGroupList( int reqId, const std::string& groups) {
    printf("Display Group List. ReqId: %d, Groups: %s\n", reqId, groups.c_str());
}
```

```
class TestWrapper(wrapper.EWrapper):
```

...

```
    def displayGroupList(self, reqId: int, groups: str):
        super().displayGroupList(reqId, groups)
        print("DisplayGroupList. Request: ", reqId, "Groups", groups)
```

21.2 Subscribe To Group Events

To integrate with a specific Group, you need to first subscribe to the group number by invoking **IBApi.EClient.↩ subscribeToGroupEvents** (p. **??**).

```
            client.subscribeToGroupEvents(9002, 1);
```

```
        client.subscribeToGroupEvents(9002, 1);
```

```
        client.subscribeToGroupEvents(9002, 1)
```

```
    m_pClient->subscribeToGroupEvents(9002, 1);
```

```
        self.subscribeToGroupEvents(19002, 1)
```

The **IBApi.EWrapper.displayGroupUpdated** call back is triggered once after receiving the subscription request, and will be sent again if the selected contract in the subscribed display group has changed.

```
    public class EWrapperImpl : EWrapper
    {
```

...

```
        public virtual void displayGroupUpdated(int reqId, string contractInfo)
        {
            Console.WriteLine("displayGroupUpdated. Request: " + reqId + ", ContractInfo: " + contractInfo)
    ;
        }
```

```
public class EWrapperImpl implements EWrapper {
```

...

```
    @Override
    public void displayGroupUpdated(int reqId, String contractInfo) {
        System.out.println("Display Group Updated. ReqId: "+reqId+", Contract info: "+contractInfo+"\n");
    }
```

```
        Public Sub displayGroupUpdated(reqId As Integer, contractInfo As String) Implements
    IBApi.EWrapper.displayGroupUpdated
        Console.WriteLine("DisplayGroupUpdated - ReqId [" & reqId & "] ContractInfo [" & contractInfo &
    "]")
    End Sub
```

- ```
  class TestCppClient : public EWrapper
  {
  ```

  ...

  ```
  void TestCppClient::displayGroupUpdated( int reqId, const std::string& contractInfo) {
      std::cout << "Display Group Updated. ReqId: " << reqId << ", Contract Info: " << contractInfo <<
        std::endl;
  }
  ```

- ```
  class TestWrapper(wrapper.EWrapper):
  ```

  ...

  ```
      def displayGroupUpdated(self, reqId: int, contractInfo: str):
          super().displayGroupUpdated(reqId, contractInfo)
          print("displayGroupUpdated. Request:", reqId, "ContractInfo:", contractInfo)
  ```

21.3 Update Display Group

Once you have subscribed to a specific Group, you can then have the Group Window in TWS to display a certain contract by invoking **IBApi.EClient.updateDisplayGroup** (p. **??**).

The encoded value that uniquely represents the contract in IB. Possible values include:

1. none = empty selection
2. contractID@exchange - any non-combination contract. Examples: 8314@SMART for IBM SMART; 8314@ARCA for IBM @ARCA
3. combo = if any combo is selected

- ```
              client.updateDisplayGroup(9002, "8314@SMART");
  ```
- ```
          client.updateDisplayGroup(9002, "8314@SMART");
  ```
- ```
          client.updateDisplayGroup(9002, "8314@SMART")
  ```
- ```
      m_pClient->updateDisplayGroup(9002, "8314@SMART");
  ```
- ```
          self.updateDisplayGroup(19002, "8314@SMART")
  ```

Note: This request from the API does not get a response from TWS unless an error occurs.

In the above sample we have commanded TWS Windows that chained with Group #1 to display IBM@SMART. The screenshot of TWS Mosaic below shows that both the pink chained (Group #1) windows are now displaying IBM@SMART, while the green chained (Group #4) window remains unchanged:

21.4 Unsubscribe From Group Events

Invoke the **IBApi.EClient.unsubscribeFromGroupEvents** method to cancel a group subscription.

- ```
              client.unsubscribeFromGroupEvents(9002);
  ```
- ```
          client.unsubscribeFromGroupEvents(9002);
  ```
- ```
          client.unsubscribeFromGroupEvents(9002)
  ```
- ```
      m_pClient->unsubscribeFromGroupEvents(9002);
  ```
- ```
          self.unsubscribeFromGroupEvents(19002)
  ```

Chapter 21

Namespace Index

22.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 22

Hierarchical Index

http://interactivebrokers.github.io/tws-api/hierarchy.html

23.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 23

Class Index

http://interactivebrokers.github.io/tws-api/classes.html

24.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 24

Namespace Documentation

25.1 IBApi Namespace Reference

Classes

- class **AccountSummaryTags**

  *class containing all existing values being reported by **EClientSocket::reqAccountSummary***
- class **Bar**

  The historical data bar's description.
- class **CodeMsgPair**

  associates error code and error message as a pair.
- class **ComboLeg**

  Class representing a leg within combo orders.
- class **CommissionReport**

  class representing the commissions generated by an execution.
- class **Contract**

  class describing an instrument's definition
- class **ContractDetails**

  extended contract details.
- class **CTriggerMethod**
- class **DeltaNeutralContract**

  *Delta-Neutral **Contract** (*p. **??**).*
- class **EClient**

  TWS/Gateway client class This client class contains all the available methods to communicate with IB. Up to thirty-two clients can be connected to a single instance of the TWS/Gateway simultaneously. From herein, the TWS/Gateway will be referred to as the Host.
- class **EClientErrors**

  Contains all possible errors occurring on the client side. This errors are not sent by the TWS but rather generated as the result of malfunction within the TWS API client.
- class **EClientSocket**

  TWS/Gateway client class This client class contains all the available methods to communicate with IB. Up to 32 clients can be connected to a single instance of the TWS/Gateway simultaneously. From herein, the TWS/Gateway will be referred to as the Host.
- class **EClientSocketSSL**

  *Implements a Secure Socket Layer (SSL) on top of the **EClientSocket** class.*
- class **EReader**

  Captures incoming messages to the API client and places them into a queue.

- interface **EReaderSignal**

  Notifies the thread reading information from the TWS whenever there are messages ready to be consumed. Not currently used in Python API.
- interface **EWrapper**

  *This interface's methods are used by the TWS/Gateway to communicate with the API client. Every API client application needs to implement this interface in order to handle all the events generated by the TWS/Gateway. Almost every **EClientSocket** method call will result in at least one event delivered here.*
- class **Execution**

  Class describing an order's execution.
- class **ExecutionCondition**

  *This class represents a condition requiring a specific execution event to be fulfilled. Orders can be activated or canceled if a set of given conditions is met. An **ExecutionCondition** is met whenever a trade occurs on a certain product at the given exchange.*
- class **ExecutionFilter**

  when requesting executions, a filter can be specified to receive only a subset of them
- class **HistoricalTick**

  The historical tick's description. Used when requesting historical tick data with whatToShow = MIDPOINT.
- class **HistoricalTickBidAsk**

  The historical tick's description. Used when requesting historical tick data with whatToShow = BID_ASK.
- class **HistoricalTickLast**

  The historical tick's description. Used when requesting historical tick data with whatToShow = TRADES.
- class **Liquidity**

  Class describing the liquidity type of an execution.
- class **Order**

  The order's description.
- class **OrderComboLeg**

  Allows to specify a price on an order's leg.
- class **OrderState**

  Provides an active order's current state.
- class **PercentChangeCondition**

  Used with conditional orders to place or submit an order based on a percentage change of an instrument to the last close price.
- class **PriceCondition**

  Used with conditional orders to cancel or submit order based on price of an instrument.
- class **ScannerSubscription**

  Defines a market scanner request.
- class **SoftDollarTier**

  A container for storing Soft Dollar Tier information.
- class **TagValue**

  Convenience class to define key-value pairs.
- class **TickAttrib**

  Tick attributes that describes additional information for price ticks.
- class **TimeCondition**

  Time condition used in conditional orders to submit or cancel orders at specified time.
- class **VolumeCondition**

  Used with conditional orders to submit or cancel an order based on a specified volume change in a security.

Enumerations

- enum **TriggerMethod** {
  Default = 0, **DoubleBidAsk**, **Last**, **DoubleLast**,
  BidAsk, **LastOfBidAsk** = 7, **MidPoint** }

Chapter 25

Class Documentation

26.1 AccountSummaryTags Class Reference

class containing all existing values being reported by **EClientSocket::reqAccountSummary**

Static Public Member Functions

- static string **GetAllTags** ()

Public Attributes

- const string **AccountType** = "AccountType"
- const string **NetLiquidation** = "NetLiquidation"
- const string **TotalCashValue** = "TotalCashValue"
- const string **SettledCash** = "SettledCash"
- const string **AccruedCash** = "AccruedCash"
- const string **BuyingPower** = "BuyingPower"
- const string **EquityWithLoanValue** = "EquityWithLoanValue"
- const string **PreviousEquityWithLoanValue** = "PreviousEquityWithLoanValue"
- const string **GrossPositionValue** = "GrossPositionValue"
- const string **ReqTEquity** = "ReqTEquity"
- const string **ReqTMargin** = "ReqTMargin"
- const string **SMA** = "SMA"
- const string **InitMarginReq** = "InitMarginReq"
- const string **MaintMarginReq** = "MaintMarginReq"
- const string **AvailableFunds** = "AvailableFunds"
- const string **ExcessLiquidity** = "ExcessLiquidity"
- const string **Cushion** = "Cushion"
- const string **FullInitMarginReq** = "FullInitMarginReq"
- const string **FullMaintMarginReq** = "FullMaintMarginReq"
- const string **FullAvailableFunds** = "FullAvailableFunds"
- const string **FullExcessLiquidity** = "FullExcessLiquidity"
- const string **LookAheadNextChange** = "LookAheadNextChange"
- const string **LookAheadInitMarginReq** = "LookAheadInitMarginReq"
- const string **LookAheadMaintMarginReq** = "LookAheadMaintMarginReq"
- const string **LookAheadAvailableFunds** = "LookAheadAvailableFunds"
- const string **LookAheadExcessLiquidity** = "LookAheadExcessLiquidity"
- const string **HighestSeverity** = "HighestSeverity"
- const string **DayTradesRemaining** = "DayTradesRemaining"
- const string **Leverage** = "Leverage"

26.1.1 Detailed Description

class containing all existing values being reported by **EClientSocket::reqAccountSummary**

26.2 Bar Class Reference

The historical data bar's description.

Public Member Functions

- **Bar** (string time, double open, double high, double low, double close, long volume, int count, double wap)

Properties

- string **Time** `[get]`

  The bar's date and time (either as a yyyymmss hh:mm:ss formatted string or as system time according to the request). Time zone is the TWS time zone chosen on login.
- double **Open** `[get]`

  The bar's open price.
- double **High** `[get]`

  The bar's high price.
- double **Low** `[get]`

  The bar's low price.
- double **Close** `[get]`

  The bar's close price.
- long **Volume** `[get]`

  The bar's traded volume if available (only available for TRADES)
- double **WAP** `[get]`

  The bar's Weighted Average Price (only available for TRADES)
- int **Count** `[get]`

  The number of trades during the bar's timespan (only available for TRADES)

26.2.1 Detailed Description

The historical data bar's description.

See also

EClient (p. **??**), **EWrapper**

26.3 CodeMsgPair Class Reference

associates error code and error message as a pair.

Public Member Functions

- **CodeMsgPair** (int code, string message)

Properties

- int **Code** [get]
- string **Message** [get]

26.3.1 Detailed Description

associates error code and error message as a pair.

26.4 ComboLeg Class Reference

Class representing a leg within combo orders.

Public Member Functions

- **ComboLeg** (int conId, int ratio, string action, string exchange, int openClose, int shortSaleSlot, string designatedLocation, int exemptCode)

Static Public Attributes

- static int **SAME** = 0
- static int **OPEN** = 1
- static int **CLOSE** = 2
- static int **UNKNOWN** = 3

Properties

- int **ConId** [get, set]

  *The **Contract** (p. **??**)'s IB's unique id.*
- int **Ratio** [get, set]

  Select the relative number of contracts for the leg you are constructing. To help determine the ratio for a specific combination order, refer to the Interactive Analytics section of the User's Guide.
- string **Action** [get, set]

  The side (buy or sell) of the leg:

  .
- string **Exchange** [get, set]

  The destination exchange to which the order will be routed.
- int **OpenClose** [get, set]

  Specifies whether an order is an open or closing order. For institutional customers to determine if this order is to open or close a position. 0 - Same as the parent security. This is the only option for retail customers.
  1 - Open. This value is only valid for institutional customers.
  2 - Close. This value is only valid for institutional customers.
  3 - Unknown.
- int **ShortSaleSlot** [get, set]

  For stock legs when doing short selling. Set to 1 = clearing broker, 2 = third party.
- string **DesignatedLocation** [get, set]

  When ShortSaleSlot is 2, this field shall contain the designated location.
- int **ExemptCode** [get, set]

  DOC_TODO.

26.4.1 Detailed Description

Class representing a leg within combo orders.

See also

Order

26.4.2 Property Documentation

26.4.2.1 Action

```
string Action  [get], [set]
```

The side (buy or sell) of the leg:
.

- For individual accounts, only BUY and SELL are available. SSHORT is for institutions.

26.5 CommissionReport Class Reference

class representing the commissions generated by an execution.

Public Member Functions

- override bool **Equals** (Object p_other)

Properties

- string **ExecId** `[get, set]`
  the execution's id this commission belongs to.
- double **Commission** `[get, set]`
  the commissions cost.
- string **Currency** `[get, set]`
  the reporting currency.
- double **RealizedPNL** `[get, set]`
  the realized profit and loss
- double **Yield** `[get, set]`
  The income return.
- int **YieldRedemptionDate** `[get, set]`
  date expressed in yyyymmdd format.

26.5.1 Detailed Description

class representing the commissions generated by an execution.

See also

Execution

26.6 Contract Class Reference

class describing an instrument's definition

Public Member Functions

- override string **ToString** ()

Properties

- int **ConId** `[get, set]`

  The unique IB contract identifier.
- string **Symbol** `[get, set]`

  The underlying's asset symbol.
- string **SecType** `[get, set]`

  The security's type: STK - stock (or ETF) OPT - option FUT - future IND - index FOP - futures option CASH - forex pair BAG - combo WAR - warrant BOND- bond CMDTY- commodity NEWS- news FUND- mutual fund.
- string **LastTradeDateOrContractMonth** `[get, set]`

  *The contract's last trading day or contract month (for Options and Futures). Strings with format YYYYMM will be interpreted as the **Contract** Month whereas YYYYMMDD will be interpreted as Last Trading Day.*
- double **Strike** `[get, set]`

  The option's strike price.
- string **Right** `[get, set]`

  Either Put or Call (i.e. Options). Valid values are P, PUT, C, CALL.
- string **Multiplier** `[get, set]`

  The instrument's multiplier (i.e. options, futures).
- string **Exchange** `[get, set]`

  The destination exchange.
- string **Currency** `[get, set]`

  The underlying's currency.
- string **LocalSymbol** `[get, set]`

  The contract's symbol within its primary exchange For options, this will be the OCC symbol.
- string **PrimaryExch** `[get, set]`

  The contract's primary exchange. For smart routed contracts, used to define contract in case of ambiguity. Should be defined as native exchange of contract, e.g. ISLAND for MSFT For exchanges which contain a period in name, will only be part of exchange name prior to period, i.e. ENEXT for ENEXT.BE.
- string **TradingClass** `[get, set]`

  The trading class name for this contract. Available in TWS contract description window as well. For example, GBL Dec '13 future's trading class is "FGBL".
- bool **IncludeExpired** `[get, set]`

If set to true, contract details requests and historical data queries can be performed pertaining to expired futures contracts. Expired options or other instrument types are not available.

- string **SecIdType** `[get, set]`

  Security's identifier when querying contract's details or placing orders ISIN - Example: Apple: US0378331005 CUSIP - Example: Apple: 037833100.

- string **SecId** `[get, set]`

  Identifier of the security type.

- string **ComboLegsDescription** `[get, set]`

  Description of the combo legs.

- List< **ComboLeg** > **ComboLegs** `[get, set]`

  The legs of a combined contract definition.

- **DeltaNeutralContract DeltaNeutralContract** `[get, set]`

  Delta and underlying price for Delta-Neutral combo orders. Underlying (STK or FUT), delta and underlying price goes into this attribute.

26.6.1 Detailed Description

class describing an instrument's definition

See also

ContractDetails

26.6.2 Property Documentation

26.6.2.1 ComboLegs

`List< ComboLeg> ComboLegs [get], [set]`

The legs of a combined contract definition.

See also

ComboLeg

26.6.2.2 DeltaNeutralContract

`DeltaNeutralContract DeltaNeutralContract [get], [set]`

Delta and underlying price for Delta-Neutral combo orders. Underlying (STK or FUT), delta and underlying price goes into this attribute.

See also

DeltaNeutralContract

26.6.2.3 SecId

```
string SecId  [get], [set]
```

Identifier of the security type.

See also

secIdType

26.7 ContractDetails Class Reference

extended contract details.

Public Member Functions

- **ContractDetails** (**Contract** summary, String marketName, double minTick, String orderTypes, String validExchanges, int underConId, String longName, String contractMonth, String industry, String category, String subcategory, String timeZoneId, String tradingHours, String liquidHours, String evRule, double evMultiplier, int aggGroup)

Properties

- **Contract Contract**

  *A fully-defined **Contract** object.*
- string **MarketName**

  The market name for this product.
- double **MinTick**

  *The minimum allowed price variation. Note that many securities vary their minimum tick size according to their price. This value will only show the smallest of the different minimum tick sizes regardless of the product's price. Full information about the minimum increment price structure can be obtained with the reqMarketRule function or the IB **Contract** and Security Search site.*
- int **PriceMagnifier**

  Allows execution and strike prices to be reported consistently with market data, historical data and the order price, i.e. Z on LIFFE is reported in Index points and not GBP.
- string **OrderTypes**

  Supported order types for this product.
- string **ValidExchanges**

  Valid exchange fields when placing an order for this contract.
  The list of exchanges will is provided in the same order as the corresponding MarketRuleIds list.
- int **UnderConId**

  For derivatives, the contract ID (conID) of the underlying instrument.
- string **LongName**

  Descriptive name of the product.
- string **ContractMonth**

  Typically the contract month of the underlying for a Future contract.
- string **Industry**

  The industry classification of the underlying/product. For example, Financial.
- string **Category**

  The industry category of the underlying. For example, InvestmentSvc.

- string **Subcategory**

  The industry subcategory of the underlying. For example, Brokerage.
- string **TimeZoneId**

  The time zone for the trading hours of the product. For example, EST.
- string **TradingHours**

  The trading hours of the product. This value will contain the trading hours of the current day as well as the next's. For example, 20090507:0700-1830,1830-2330;20090508:CLOSED. In TWS versions 965+ there is an option in the Global Configuration API settings to return 1 month of trading hours. In TWS version 970+, the format includes the date of the closing time to clarify potential ambiguity, ex: 20180323:0400-20180323:2000;20180326:0400-20180326:2000 The trading hours will correspond to the hours for the product on the associated exchange. The same instrument can have different hours on different exchanges.
- string **LiquidHours**

  The liquid hours of the product. This value will contain the liquid hours (regular trading hours) of the contract on the specified exchange. Format for TWS versions until 969: 20090507:0700-1830,1830-2330;20090508:CLOS↩ ED. In TWS versions 965+ there is an option in the Global Configuration API settings to return 1 month of trading hours. In TWS v970 and above, the format includes the date of the closing time to clarify potential ambiguity, e.g. 20180323:0930-20180323:1600;20180326:0930-20180326:1600.
- string **EvRule**

  Contains the Economic Value Rule name and the respective optional argument. The two values should be separated by a colon. For example, aussieBond:YearsToExpiration=3. When the optional argument is not present, the first value will be followed by a colon.
- double **EvMultiplier**

  Tells you approximately how much the market value of a contract would change if the price were to change by 1. It cannot be used to get market value by multiplying the price by the approximate multiplier.
- int **MdSizeMultiplier**

  MD Size Multiplier. Returns the size multiplier for values returned to tickSize from a market data request. Generally 100 for US stocks and 1 for other instruments.
- int **AggGroup**

  Aggregated group Indicates the smart-routing group to which a contract belongs. contracts which cannot be smart-routed have aggGroup = -1.
- List< **TagValue** > **SecIdList**

  A list of contract identifiers that the customer is allowed to view. CUSIP/ISIN/etc. For US stocks, receiving the ISIN requires the CUSIP market data subscription. For Bonds, the CUSIP or ISIN is input directly into the symbol field of the ***Contract*** *class.*

- string **UnderSymbol**

  For derivatives, the symbol of the underlying contract.
- string **UnderSecType**

  For derivatives, returns the underlying security type.
- string **MarketRuleIds**

  The list of market rule IDs separated by comma Market rule IDs can be used to determine the minimum price increment at a given price.
- string **RealExpirationDate**

  Real expiration date. Requires TWS 968+ and API v973.04+. Python API specifically requires API v973.06+.
- string **LastTradeTime**

  Last trade time.
- string **Cusip**

  The nine-character bond CUSIP. For Bonds only. Receiving CUSIPs requires a CUSIP market data subscription.
- string **Ratings**

  Identifies the credit rating of the issuer. This field is not currently available from the TWS API. For Bonds only. A higher credit rating generally indicates a less risky investment. Bond ratings are from Moody's and S&P respectively. Not currently implemented due to bond market data restrictions.
- string **DescAppend**

  A description string containing further descriptive information about the bond. For Bonds only.
- string **BondType**

The type of bond, such as "CORP.".

- string **CouponType**

  The type of bond coupon. This field is currently not available from the TWS API. For Bonds only.
- bool **Callable**

  If true, the bond can be called by the issuer under certain conditions. This field is currently not available from the TWS API. For Bonds only.
- bool **Putable**

  Values are True or False. If true, the bond can be sold back to the issuer under certain conditions. This field is currently not available from the TWS API. For Bonds only.
- double **Coupon**

  The interest rate used to calculate the amount you will receive in interest payments over the course of the year. This field is currently not available from the TWS API. For Bonds only.
- bool **Convertible**

  Values are True or False. If true, the bond can be converted to stock under certain conditions. This field is currently not available from the TWS API. For Bonds only.
- string **Maturity**

  he date on which the issuer must repay the face value of the bond. This field is currently not available from the TWS API. For Bonds only. Not currently implemented due to bond market data restrictions.
- string **IssueDate**

  The date the bond was issued. This field is currently not available from the TWS API. For Bonds only. Not currently implemented due to bond market data restrictions.
- string **NextOptionDate**

  Only if bond has embedded options. This field is currently not available from the TWS API. Refers to callable bonds and puttable bonds. Available in TWS description window for bonds.
- string **NextOptionType**

  Type of embedded option. This field is currently not available from the TWS API. Only if bond has embedded options.
- bool **NextOptionPartial**

  Only if bond has embedded options. This field is currently not available from the TWS API. For Bonds only.
- string **Notes**

  If populated for the bond in IB's database. For Bonds only.

26.7.1 Detailed Description

extended contract details.

See also

Contract

26.8 DeltaNeutralContract Class Reference

Delta-Neutral **Contract** (p. **??**).

Properties

- int **ConId** `[get, set]`

  The unique contract identifier specifying the security. Used for Delta-Neutral Combo contracts.
- double **Delta** `[get, set]`

  The underlying stock or future delta. Used for Delta-Neutral Combo contracts.
- double **Price** `[get, set]`

  The price of the underlying. Used for Delta-Neutral Combo contracts.

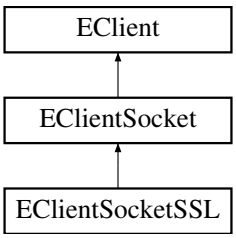
26.8.1 Detailed Description

Delta-Neutral **Contract** (p. **??**).

26.9 EClient Class Reference

TWS/Gateway client class This client class contains all the available methods to communicate with IB. Up to thirty-two clients can be connected to a single instance of the TWS/Gateway simultaneously. From herein, the TWS/↩ Gateway will be referred to as the Host.

Inheritance diagram for EClient:



Public Member Functions

- **EClient** (**EWrapper** wrapper)

  Constructor.
- void **SetConnectOptions** (string connectOptions)

  Ignore. Used for IB's internal purposes.
- void **DisableUseV100Plus** ()

  Allows to switch between different current (V100+) and previous connection mechanisms.
- bool **IsConnected** ()

  Indicates whether the API-TWS connection has been closed. Note: This function is not automatically invoked and must be by the API client.
- void **startApi** ()

  Initiates the message exchange between the client application and the TWS/IB Gateway.
- void **Close** ()

  *Terminates the connection and notifies the **EWrapper** implementing class.*
- virtual void **eDisconnect** (bool resetState=true)

  Closes the socket connection and terminates its thread.
- void **cancelTickByTickData** (int requestId)

  Cancels tick-by-tick data.

  .
- void **reqTickByTickData** (int requestId, **Contract** contract, string tickType, int numberOfTicks, bool ignore↩ Size)

  Requests tick-by-tick data.

  .
- void **cancelHistoricalData** (int reqId)

  Cancels a historical data request.
- void **calculateImpliedVolatility** (int reqId, **Contract** contract, double optionPrice, double underPrice, List< **TagValue** > impliedVolatilityOptions)

Calculate the volatility for an option.
Request the calculation of the implied volatility based on hypothetical option and its underlying prices.
*The calculation will be return in **EWrapper** (*p. **??**)'s tickOptionComputation callback.*
.

- void **calculateOptionPrice** (int reqId, **Contract** contract, double volatility, double underPrice, List< **TagValue** > optionPriceOptions)

  Calculates an option's price based on the provided volatility and its underlying's price.
  *The calculation will be return in **EWrapper** (*p. **??**)'s tickOptionComputation callback.*
  .
- void **cancelAccountSummary** (int reqId)

  Cancels the account's summary request. After requesting an account's summary, invoke this function to cancel it.
- void **cancelCalculateImpliedVolatility** (int reqId)

  Cancels an option's implied volatility calculation request.
- void **cancelCalculateOptionPrice** (int reqId)

  Cancels an option's price calculation request.
- void **cancelFundamentalData** (int reqId)

  Cancels Fundamental data request.
- void **cancelMktData** (int tickerId)

  Cancels a RT Market Data request.
- void **cancelMktDepth** (int tickerId)

  Cancel's market depth's request.
- void **cancelNewsBulletin** ()

  Cancels IB's news bulletin subscription.
- void **cancelOrder** (int orderId)

  Cancels an active order placed by from the same API client ID.
  Note: API clients cannot cancel individual orders placed by other clients. Only reqGlobalCancel is available.
  .
- void **cancelPositions** ()

  Cancels a previous position subscription request made with reqPositions.
- void **cancelRealTimeBars** (int tickerId)

  Cancels Real Time Bars' subscription.
- void **cancelScannerSubscription** (int tickerId)

  Cancels Scanner Subscription.
- void **exerciseOptions** (int tickerId, **Contract** contract, int exerciseAction, int exerciseQuantity, string account, int ovrd)

  Exercises an options contract
  Note: this function is affected by a TWS setting which specifies if an exercise request must be finalized.
- void **placeOrder** (int id, **Contract** contract, **Order** order)

  Places or modifies an order.
- void **replaceFA** (int faDataType, string xml)

  Replaces Financial Advisor's settings A Financial Advisor can define three different configurations:
- void **requestFA** (int faDataType)

  Requests the FA configuration A Financial Advisor can define three different configurations:
- void **reqAccountSummary** (int reqId, string group, string tags)

  Requests a specific account's summary.
  *This method will subscribe to the account summary as presented in the TWS' Account Summary tab. The data is returned at **EWrapper::accountSummary***
  `https://www.interactivebrokers.com/en/software/tws/accountwindowtop.htm`.
- void **reqAccountUpdates** (bool subscribe, string acctCode)

  *Subscribes to an specific account's information and portfolio Through this method, a single account's subscription can be started/stopped. As a result from the subscription, the account's information, portfolio and last update time will be received at **EWrapper::updateAccountValue** (*p. **??**), EWrapper::updateAccountPortfolio, **EWrapper::updateAccountTime** respectively. All account values and positions will be returned initially, and then there will only be updates when there is a change in a position, or to an account value every 3 minutes if it has changed. Only one*

account can be subscribed at a time. A second subscription request for another account when the previous one is still active will cause the first one to be canceled in favour of the second one. Consider user reqPositions if you want to retrieve all your accounts' portfolios directly.

- void **reqAllOpenOrders** ()

  Requests all current open orders in associated accounts at the current moment. The existing orders will be received via the openOrder and orderStatus events. Open orders are returned once; this function does not initiate a subscription.

- void **reqAutoOpenOrders** (bool autoBind)

  Requests status updates about future orders placed from TWS. Can only be used with client ID 0.

- void **reqContractDetails** (int reqId, **Contract** contract)

  Requests contract information.
  *This method will provide all the contracts matching the contract provided. It can also be used to retrieve complete options and futures chains. This information will be returned at **EWrapper** (*p. **??***):contractDetails. Though it is now (in API version > 9.72.12) advised to use reqSecDefOptParams for that purpose.*
  .

- void **reqCurrentTime** ()

  Requests TWS's current time.

- void **reqExecutions** (int reqId, **ExecutionFilter** filter)

  *Requests current day's (since midnight) executions matching the filter. Only the current day's executions can be retrieved. Along with the executions, the **CommissionReport** will also be returned. The execution details will arrive at **EWrapper** (*p. **??***):execDetails.*

- void **reqFundamentalData** (int reqId, **Contract** contract, String reportType, List< **TagValue** > fundamentalDataOptions)

  *Requests the contract's Reuters or Wall Street Horizons fundamental data. Fundalmental data is returned at **EWrapper::fundamentalData** (*p. **??***).*

- void **reqGlobalCancel** ()

  Cancels all active orders.
  This method will cancel ALL open orders including those placed directly from TWS.

- void **reqHistoricalData** (int tickerId, **Contract** contract, string endDateTime, string durationString, string barSizeSetting, string whatToShow, int useRTH, int formatDate, bool keepUpToDate, List< **TagValue** > chartOptions)

  Requests contracts' historical data. When requesting historical data, a finishing time and date is required along with a duration string. For example, having:

- void **reqIds** (int numIds)

  Requests the next valid order ID at the current moment.

- void **reqManagedAccts** ()

  Requests the accounts to which the logged user has access to.

- void **reqMktData** (int tickerId, **Contract** contract, string genericTickList, bool snapshot, bool regulatory Snaphsot, List< **TagValue** > mktDataOptions)

  Requests real time market data. Returns market data for an instrument either in real time or 10-15 minutes delayed (depending on the market data type specified)

- void **reqMarketDataType** (int marketDataType)

  Switches data type returned from reqMktData request to "frozen", "delayed" or "delayed-frozen" market data. Requires TWS/IBG v963+.
  The API can receive frozen market data from Trader Workstation. Frozen market data is the last data recorded in our system.
  During normal trading hours, the API receives real-time market data. Invoking this function with argument 2 requests a switch to frozen data immediately or after the close.
  When the market reopens the next data the market data type will automatically switch back to real time if available.

- void **reqMarketDepth** (int tickerId, **Contract** contract, int numRows, List< **TagValue** > mktDepthOptions)

  Requests the contract's market depth (order book).
  This request must be direct-routed to an exchange and not smart-routed. The number of simultaneous market depth requests allowed in an account is calculated based on a formula that looks at an accounts equity, commissions, and quote booster packs.

- void **reqNewsBulletins** (bool allMessages)

Subscribes to IB's News Bulletins.

- void **reqOpenOrders** ()

  Requests all open orders places by this specific API client (identified by the API client id). For client ID 0, this will bind previous manual TWS orders.

- void **reqPositions** ()

  Subscribes to position updates for all accessible accounts. All positions sent initially, and then only updates as positions change.

- void **reqRealTimeBars** (int tickerId, **Contract** contract, int barSize, string whatToShow, bool useRTH, List< **TagValue** > realTimeBarsOptions)

  Requests real time bars
  Currently, only 5 seconds bars are provided. This request is subject to the same pacing as any historical data request: no more than 60 API queries in more than 600 seconds.
  Real time bars subscriptions are also included in the calculation of the number of Level 1 market data subscriptions allowed in an account.

- void **reqScannerParameters** ()

  Requests an XML list of scanner parameters valid in TWS.
  Not all parameters are valid from API scanner.

- void **reqScannerSubscription** (int reqId, **ScannerSubscription** subscription, List< **TagValue** > scanner↩ SubscriptionOptions)

  Starts a subscription to market scan results based on the provided parameters.

- void **setServerLogLevel** (int logLevel)

  Changes the TWS/GW log level. The default is 2 = ERROR
  5 = DETAIL is required for capturing all API messages and troubleshooting API programs
  Valid values are:
  1 = SYSTEM
  2 = ERROR
  3 = WARNING
  4 = INFORMATION
  5 = DETAIL
  .

- void **verifyRequest** (string apiName, string apiVersion)

  For IB's internal purpose. Allows to provide means of verification between the TWS and third party programs.

- void **verifyMessage** (string apiData)

  For IB's internal purpose. Allows to provide means of verification between the TWS and third party programs.

- void **verifyAndAuthRequest** (string apiName, string apiVersion, string opaqueIsvKey)

  For IB's internal purpose. Allows to provide means of verification between the TWS and third party programs.

- void **verifyAndAuthMessage** (string apiData, string xyzResponse)

  For IB's internal purpose. Allows to provide means of verification between the TWS and third party programs.

- void **queryDisplayGroups** (int requestId)

  Requests all available Display Groups in TWS.

- void **subscribeToGroupEvents** (int requestId, int groupId)

  Integrates API client and TWS window grouping.

- void **updateDisplayGroup** (int requestId, string contractInfo)

  Updates the contract displayed in a TWS Window Group.

- void **unsubscribeFromGroupEvents** (int requestId)

  Cancels a TWS Window Group subscription.

- void **reqPositionsMulti** (int requestId, string account, string modelCode)

  Requests position subscription for account and/or model Initially all positions are returned, and then updates are returned for any position changes in real time.

- void **cancelPositionsMulti** (int requestId)

  Cancels positions request for account and/or model.

- void **reqAccountUpdatesMulti** (int requestId, string account, string modelCode, bool ledgerAndNLV)

  Requests account updates for account and/or model.

- void **cancelAccountUpdatesMulti** (int requestId)

Cancels account updates request for account and/or model.

- void **reqSecDefOptParams** (int reqId, string underlyingSymbol, string futFopExchange, string underlying↩ SecType, int underlyingConId)

  Requests security definition option parameters for viewing a contract's option chain.
- void **reqSoftDollarTiers** (int reqId)

  Requests pre-defined Soft Dollar Tiers. This is only supported for registered professional advisors and hedge and mutual funds who have configured Soft Dollar Tiers in Account Management. Refer to: `https://www.↩ interactivebrokers.com/en/software/am/am/manageaccount/requestsoftdollars.↩ htm?Highlight=soft%20dollar%20tier`.
- void **reqFamilyCodes** ()

  Requests family codes for an account, for instance if it is a FA, IBroker, or associated account.
- void **reqMatchingSymbols** (int reqId, string pattern)

  Requests matching stock symbols.
- void **reqMktDepthExchanges** ()

  Requests venues for which market data is returned to updateMktDepthL2 (those with market makers)
- void **reqSmartComponents** (int reqId, String bboExchange)

  Returns the mapping of single letter codes to exchange names given the mapping identifier.
- void **reqNewsProviders** ()

  Requests news providers which the user has subscribed to.
- void **reqNewsArticle** (int requestId, string providerCode, string articleId, List< **TagValue** > newsArticle↩ Options)

  Requests news article body given articleId.
- void **reqHistoricalNews** (int requestId, int conId, string providerCodes, string startDateTime, string end↩ DateTime, int totalResults, List< **TagValue** > historicalNewsOptions)

  Requests historical news headlines.
- void **reqHeadTimestamp** (int tickerId, **Contract** contract, string whatToShow, int useRTH, int formatDate)

  Returns the timestamp of earliest available historical data for a contract and data type.
- void **cancelHeadTimestamp** (int tickerId)

  Cancels a pending reqHeadTimeStamp request

  .
- void **reqHistogramData** (int tickerId, **Contract** contract, bool useRTH, string period)

  Returns data histogram of specified contract

  .
- void **cancelHistogramData** (int tickerId)

  Cancels an active data histogram request.
- void **reqMarketRule** (int marketRuleId)

  Requests details about a given market rule
  The market rule for an instrument on a particular exchange provides details about how the minimum price increment changes with price
  A list of market rule ids can be obtained by invoking reqContractDetails on a particular contract. The returned market rule ID list will provide the market rule ID for the instrument in the correspond valid exchange list in contractDetails.

  .
- void **reqPnL** (int reqId, string account, string modelCode)

  Creates subscription for real time daily PnL and unrealized PnL updates.
- void **cancelPnL** (int reqId)

  cancels subscription for real time updated daily PnL params reqId
- void **reqPnLSingle** (int reqId, string account, string modelCode, int conId)

  Requests real time updates for daily PnL of individual positions.
- void **cancelPnLSingle** (int reqId)

  Cancels real time subscription for a positions daily PnL information.
- void **reqHistoricalTicks** (int reqId, **Contract** contract, string startDateTime, string endDateTime, int numberOfTicks, string whatToShow, int useRth, bool ignoreSize, List< **TagValue** > miscOptions)

  Requests historical Time&Sales data for an instrument.
- int **ReadInt** ()
- byte [ ] **ReadAtLeastNBytes** (int msgSize)
- byte [ ] **ReadByteArray** (int msgSize)

Protected Member Functions

- abstract uint **prepareBuffer** (BinaryWriter paramsList)
- void **sendConnectRequest** ()
- bool **CheckServerVersion** (int requiredVersion)
- bool **CheckServerVersion** (int requestId, int requiredVersion)
- bool **CheckServerVersion** (int requiredVersion, string updatetail)
- bool **CheckServerVersion** (int tickerId, int requiredVersion, string updatetail)
- void **CloseAndSend** (BinaryWriter paramsList, uint lengthPos, **CodeMsgPair** error)
- void **CloseAndSend** (int reqId, BinaryWriter paramsList, uint lengthPos, **CodeMsgPair** error)
- abstract void **CloseAndSend** (BinaryWriter request, uint lengthPos)
- bool **CheckConnection** ()
- void **ReportError** (int reqId, **CodeMsgPair** error, string tail)
- void **ReportUpdateTWS** (int reqId, string tail)
- void **ReportUpdateTWS** (string tail)
- void **ReportError** (int reqId, int code, string message)
- void **SendCancelRequest** (OutgoingMessages msgType, int version, int reqId, **CodeMsgPair** error↩ Message)
- void **SendCancelRequest** (OutgoingMessages msgType, int version, **CodeMsgPair** errorMessage)
- bool **VerifyOrderContract** (**Contract** contract, int id)
- bool **VerifyOrder** (**Order** order, int id, bool isBagOrder)

Protected Attributes

- int **serverVersion**
- ETransport **socketTransport**
- **EWrapper wrapper**
- bool **isConnected**
- int **clientId**
- bool **extraAuth**
- bool **useV100Plus** = true
- bool **allowRedirect** = false
- Stream **tcpStream**

Properties

- **EWrapper Wrapper** `[get]`

  *Reference to the **EWrapper** implementing object.*
- bool **AllowRedirect** `[get, set]`
- int **ServerVersion** `[get]`

  returns the Host's version. Some of the API functionality might not be available in older Hosts and therefore it is essential to keep the TWS/Gateway as up to date as possible.
- string **ServerTime** `[get, protected set]`
- string **optionalCapabilities** `[get, set]`
- bool **AsyncEConnect** `[get, set]`

26.9.1 Detailed Description

TWS/Gateway client class This client class contains all the available methods to communicate with IB. Up to thirty-two clients can be connected to a single instance of the TWS/Gateway simultaneously. From herein, the TWS/↩ Gateway will be referred to as the Host.

26.9.2 Constructor & Destructor Documentation

26.9.2.1 EClient()

```
EClient (
            EWrapper wrapper )  [inline]
```

Constructor.

Parameters

wrapper	**EWrapper** (p. **??**)'s implementing class instance. Every message being delivered by IB to the API client will be forwarded to the **EWrapper** (p. **??**)'s implementing class.

See also

EWrapper

26.9.3 Member Function Documentation

26.9.3.1 calculateImpliedVolatility()

```
void calculateImpliedVolatility (
            int reqId,
             Contract contract,
            double optionPrice,
            double underPrice,
            List<  TagValue > impliedVolatilityOptions )  [inline]
```

Calculate the volatility for an option.
Request the calculation of the implied volatility based on hypothetical option and its underlying prices.
The calculation will be return in **EWrapper** (p. **??**)'s tickOptionComputation callback.
.

Parameters

reqId	unique identifier of the request.
contract	the option's contract for which the volatility wants to be calculated.
optionPrice	hypothetical option price.
underPrice	hypothetical option's underlying price.

See also

EWrapper::tickOptionComputation (p. **??**), **cancelCalculateImpliedVolatility** (p. **??**), **Contract**

26.9.3.2 calculateOptionPrice()

```
void calculateOptionPrice (
            int reqId,
             Contract contract,
            double volatility,
            double underPrice,
            List<  TagValue > optionPriceOptions )  [inline]
```

Calculates an option's price based on the provided volatility and its underlying's price.
The calculation will be return in **EWrapper** (p. **??**)'s tickOptionComputation callback.
.

Parameters

reqId	request's unique identifier.
contract	the option's contract for which the price wants to be calculated.
volatility	hypothetical volatility.
underPrice	hypothetical underlying's price.

See also

EWrapper::tickOptionComputation (p. **??**), **cancelCalculateOptionPrice** (p. **??**), **Contract**

26.9.3.3 cancelAccountSummary()

```
void cancelAccountSummary (
            int reqId )  [inline]
```

Cancels the account's summary request. After requesting an account's summary, invoke this function to cancel it.

Parameters

reqId	the identifier of the previously performed account request

See also

reqAccountSummary

26.9.3.4 cancelAccountUpdatesMulti()

```
void cancelAccountUpdatesMulti (
            int requestId )  [inline]
```

Cancels account updates request for account and/or model.

Parameters

requestId	account subscription to cancel

See also

reqAccountUpdatesMulti

26.9.3.5 cancelCalculateImpliedVolatility()

```
void cancelCalculateImpliedVolatility (
            int reqId )  [inline]
```

Cancels an option's implied volatility calculation request.

Parameters

reqId	the identifier of the implied volatility's calculation request.

See also

calculateImpliedVolatility

26.9.3.6 cancelCalculateOptionPrice()

```
void cancelCalculateOptionPrice (
            int reqId )  [inline]
```

Cancels an option's price calculation request.

Parameters

reqId	the identifier of the option's price's calculation request.

See also

calculateOptionPrice

26.9.3.7 cancelFundamentalData()

```
void cancelFundamentalData (
            int reqId )  [inline]
```

Cancels Fundamental data request.

Parameters

reqId	the request's identifier.

See also

reqFundamentalData

26.9.3.8 cancelHeadTimestamp()

```
void cancelHeadTimestamp (
            int tickerId )  [inline]
```

Cancels a pending reqHeadTimeStamp request
.

Parameters

tickerId	Id of the request

26.9.3.9 cancelHistogramData()

```
void cancelHistogramData (
            int tickerId )  [inline]
```

Cancels an active data histogram request.

Parameters

tickerId	- identifier specified in reqHistogramData request

See also

reqHistogramData (p. **??**), histogramData

26.9.3.10 cancelHistoricalData()

```
void cancelHistoricalData (
            int reqId )  [inline]
```

Cancels a historical data request.

Parameters

reqId	the request's identifier.

See also

reqHistoricalData

26.9.3.11 cancelMktData()

```
void cancelMktData (
            int tickerId )  [inline]
```

Cancels a RT Market Data request.

Parameters

tickerId	request's identifier

See also

reqMktData

26.9.3.12 cancelMktDepth()

```
void cancelMktDepth (
            int tickerId )  [inline]
```

Cancel's market depth's request.

Parameters

tickerId	request's identifier.

See also

reqMarketDepth

26.9.3.13 cancelNewsBulletin()

```
void cancelNewsBulletin ( )  [inline]
```

Cancels IB's news bulletin subscription.

See also

reqNewsBulletins

26.9.3.14 cancelOrder()

```
void cancelOrder (
            int orderId )  [inline]
```

Cancels an active order placed by from the same API client ID.
Note: API clients cannot cancel individual orders placed by other clients. Only reqGlobalCancel is available.
.

Parameters

orderId	the order's client id

See also

placeOrder (p. **??**), **reqGlobalCancel**

26.9.3.15 cancelPnLSingle()

```
void cancelPnLSingle (
            int reqId )  [inline]
```

Cancels real time subscription for a positions daily PnL information.

Parameters

reqId	

26.9.3.16 cancelPositions()

```
void cancelPositions ( )  [inline]
```

Cancels a previous position subscription request made with reqPositions.

See also

reqPositions

26.9.3.17 cancelPositionsMulti()

```
void cancelPositionsMulti (
            int requestId )  [inline]
```

Cancels positions request for account and/or model.

Parameters

requestId	- the identifier of the request to be canceled.

See also

reqPositionsMulti

26.9.3.18 cancelRealTimeBars()

```
void cancelRealTimeBars (
            int tickerId )  [inline]
```

Cancels Real Time Bars' subscription.

Parameters

tickerId	the request's identifier.

See also

reqRealTimeBars

26.9.3.19 cancelScannerSubscription()

```
void cancelScannerSubscription (
            int tickerId )  [inline]
```

Cancels Scanner Subscription.

Parameters

tickerId	the subscription's unique identifier.

See also

reqScannerSubscription (p. **??**), **ScannerSubscription** (p. **??**), **reqScannerParameters**

26.9.3.20 cancelTickByTickData()

```
void cancelTickByTickData (
            int requestId )  [inline]
```

Cancels tick-by-tick data.
.

Parameters

reqId	- unique identifier of the request.

26.9.3.21 Close()

```
void Close ( )  [inline]
```

Terminates the connection and notifies the **EWrapper** implementing class.

See also

EWrapper::connectionClosed (p. **??**), **eDisconnect**

26.9.3.22 exerciseOptions()

```
void exerciseOptions (
             int tickerId,
              Contract contract,
             int exerciseAction,
             int exerciseQuantity,
             string account,
             int ovrd )  [inline]
```

Exercises an options contract
Note: this function is affected by a TWS setting which specifies if an exercise request must be finalized.

Parameters

tickerId	exercise request's identifier
contract	the option **Contract** to be exercised.
exerciseAction	set to 1 to exercise the option, set to 2 to let the option lapse.
exerciseQuantity	number of contracts to be exercised
account	destination account
ovrd	Specifies whether your setting will override the system's natural action. For example, if your action is "exercise" and the option is not in-the-money, by natural action the option would not exercise. If you have override set to "yes" the natural action would be overridden and the out-of-the money option would be exercised. Set to 1 to override, set to 0 not to.

26.9.3.23 IsConnected()

```
bool IsConnected ( )  [inline]
```

Indicates whether the API-TWS connection has been closed. Note: This function is not automatically invoked and must be by the API client.

Returns

true if connection has been established, false if it has not.

26.9.3.24 placeOrder()

```
void placeOrder (
             int id,
              Contract contract,
              Order order )  [inline]
```

Places or modifies an order.

Parameters

id	the order's unique identifier. Use a sequential id starting with the id received at the nextValidId method. If a new order is placed with an order ID less than or equal to the order ID of a previous order an error will occur.
contract	the order's contract
order	the order

See also

EWrapper::nextValidId (p. **??**), **reqAllOpenOrders** (p. **??**), **reqAutoOpenOrders** (p. **??**), **reqOpenOrders** (p. **??**), **cancelOrder** (p. **??**), **reqGlobalCancel** (p. **??**), **EWrapper::openOrder** (p. **??**), **EWrapper::order↩ Status** (p. **??**), **Order** (p. **??**), **Contract**

26.9.3.25 queryDisplayGroups()

```
void queryDisplayGroups (
            int requestId )  [inline]
```

Requests all available Display Groups in TWS.

Parameters

request↩ Id	is the ID of this request

26.9.3.26 replaceFA()

```
void replaceFA (
            int faDataType,
            string xml )  [inline]
```

Replaces Financial Advisor's settings A Financial Advisor can define three different configurations:

1. Groups: offer traders a way to create a group of accounts and apply a single allocation method to all accounts in the group.
2. Profiles: let you allocate shares on an account-by-account basis using a predefined calculation value.
3. Account Aliases: let you easily identify the accounts by meaningful names rather than account numbers. More information at `https://www.interactivebrokers.com/en/?f=%2Fen%2Fsoftware%2↩ Fpdfhighlights%2FPDF-AdvisorAllocations.php%3Fib_entity%3Dllc`

Parameters

faDataType	the configuration to change. Set to 1, 2 or 3 as defined above.
xml	the xml-formatted configuration string

See also

requestFA

26.9.3.27 reqAccountSummary()

```
void reqAccountSummary (
            int reqId,
            string group,
            string tags )  [inline]
```

Requests a specific account's summary.
This method will subscribe to the account summary as presented in the TWS' Account Summary tab. The data is returned at **EWrapper::accountSummary**
https://www.interactivebrokers.com/en/software/tws/accountwindowtop.htm.

Parameters

reqId	the unique request identifier.
group	set to "All" to return account summary data for all accounts, or set to a specific Advisor Account Group name that has already been created in TWS Global Configuration.

Parameters

tags	a comma separated list with the desired tags: • AccountType — Identifies the IB account structure • NetLiquidation — The basis for determining the price of the assets in your account. Total cash value + stock value + options value + bond value • TotalCashValue — Total cash balance recognized at the time of trade + futures PNL • SettledCash — Cash recognized at the time of settlement - purchases at the time of trade - commissions - taxes - fees • AccruedCash — Total accrued cash value of stock, commodities and securities • BuyingPower — Buying power serves as a measurement of the dollar value of securities that one may purchase in a securities account without depositing additional funds • EquityWithLoanValue — Forms the basis for determining whether a client has the necessary assets to either initiate or maintain security positions. Cash + stocks + bonds + mutual funds • PreviousEquityWithLoanValue — Marginable Equity with Loan value as of 16:00 ET the previous day • GrossPositionValue — The sum of the absolute value of all stock and equity option positions • RegTEquity — Regulation T equity for universal account • RegTMargin — Regulation T margin for universal account • SMA — Special Memorandum Account: Line of credit created when the market value of securities in a Regulation T account increase in value • InitMarginReq — Initial Margin requirement of whole portfolio • MaintMarginReq — Maintenance Margin requirement of whole portfolio • AvailableFunds — This value tells what you have available for trading • ExcessLiquidity — This value shows your margin cushion, before liquidation • Cushion — Excess liquidity as a percentage of net liquidation value • FullInitMarginReq — Initial Margin of whole portfolio with no discounts or intraday credits • FullMaintMarginReq — Maintenance Margin of whole portfolio with no discounts or intraday credits • FullAvailableFunds — Available funds of whole portfolio with no discounts or intraday credits • FullExcessLiquidity — Excess liquidity of whole portfolio with no discounts or intraday credits • LookAheadNextChange — Time when look-ahead values take effect • LookAheadInitMarginReq — Initial Margin requirement of whole portfolio as of next period's margin change • LookAheadMaintMarginReq — Maintenance Margin requirement of whole portfolio as of next period's margin change • LookAheadAvailableFunds — This value reflects your available funds at the next margin change • LookAheadExcessLiquidity — This value reflects your excess liquidity at the next margin change • HighestSeverity — A measure of how close the account is to liquidation • DayTradesRemaining — The Number of Open/Close trades a user could put on before Pattern Day Trading is detected. A value of "-1" means that the user can put on unlimited day trades. • Leverage — GrossPositionValue / NetLiquidation • $LEDGER — Single flag to relay all cash balance tags∗, only in base currency. • $LEDGER:CURRENCY — Single flag to relay all cash balance tags∗, only in the specified

Parameters

See also

cancelAccountSummary (p. **??**), **EWrapper::accountSummary** (p. **??**), **EWrapper::accountSummaryEnd**

26.9.3.28 reqAccountUpdates()

```
void reqAccountUpdates (
            bool subscribe,
            string acctCode )  [inline]
```

Subscribes to an specific account's information and portfolio Through this method, a single account's subscription can be started/stopped. As a result from the subscription, the account's information, portfolio and last update time will be received at **EWrapper::updateAccountValue** (p. **??**), EWrapper::updateAccountPortfolio, **EWrapper::updateAccountTime** respectively. All account values and positions will be returned initially, and then there will only be updates when there is a change in a position, or to an account value every 3 minutes if it has changed. Only one account can be subscribed at a time. A second subscription request for another account when the previous one is still active will cause the first one to be canceled in favour of the second one. Consider user reqPositions if you want to retrieve all your accounts' portfolios directly.

Parameters

subscribe	set to true to start the subscription and to false to stop it.
acctCode	the account id (i.e. U123456) for which the information is requested.

See also

reqPositions (p. **??**), **EWrapper::updateAccountValue** (p. **??**), **EWrapper::updatePortfolio** (p. **??**), **EWrapper::updateAccountTime**

26.9.3.29 reqAccountUpdatesMulti()

```
void reqAccountUpdatesMulti (
            int requestId,
            string account,
            string modelCode,
            bool ledgerAndNLV )  [inline]
```

Requests account updates for account and/or model.

Parameters

reqId	identifier to label the request
account	account values can be requested for a particular account
modelCode	values can also be requested for a model
ledgerAndNLV	returns light-weight request; only currency positions as opposed to account values and currency positions

See also

cancelAccountUpdatesMulti (p. **??**), **EWrapper::accountUpdateMulti** (p. **??**), **EWrapper::account↩ UpdateMultiEnd**

26.9.3.30 reqAllOpenOrders()

```
void reqAllOpenOrders ( )  [inline]
```

Requests all *current* open orders in associated accounts at the current moment. The existing orders will be received via the openOrder and orderStatus events. Open orders are returned once; this function does not initiate a subscription.

See also

reqAutoOpenOrders (p. **??**), **reqOpenOrders** (p. **??**), **EWrapper::openOrder** (p. **??**), **EWrapper::order↩ Status** (p. **??**), **EWrapper::openOrderEnd**

26.9.3.31 reqAutoOpenOrders()

```
void reqAutoOpenOrders (
            bool autoBind )  [inline]
```

Requests status updates about future orders placed from TWS. Can only be used with client ID 0.

Parameters

autoBind	if set to true, the newly created orders will be assigned an API order ID and implicitly associated with this client. If set to false, future orders will not be.

See also

reqAllOpenOrders (p. **??**), **reqOpenOrders** (p. **??**), **cancelOrder** (p. **??**), **reqGlobalCancel** (p. **??**), **E↩ Wrapper::openOrder** (p. **??**), **EWrapper::orderStatus**

26.9.3.32 reqContractDetails()

```
void reqContractDetails (
            int reqId,
             Contract contract )  [inline]
```

Requests contract information.
This method will provide all the contracts matching the contract provided. It can also be used to retrieve complete options and futures chains. This information will be returned at **EWrapper** (p. **??**):contractDetails. Though it is now (in API version $>$ 9.72.12) advised to use reqSecDefOptParams for that purpose.
.

Parameters

reqId	the unique request identifier.
contract	the contract used as sample to query the available contracts. Typically, it will contain the **Contract::Symbol** (p. **??**), **Contract::Currency** (p. **??**), **Contract::SecType** (p. **??**), **Contract::Exchange**

See also

EWrapper::contractDetails (p. **??**), **EWrapper::contractDetailsEnd**

26.9.3.33 reqCurrentTime()

```
void reqCurrentTime ( )  [inline]
```

Requests TWS's current time.

See also

EWrapper::currentTime

26.9.3.34 reqExecutions()

```
void reqExecutions (
            int reqId,
             ExecutionFilter filter )  [inline]
```

Requests current day's (since midnight) executions matching the filter. Only the current day's executions can be retrieved. Along with the executions, the **CommissionReport** will also be returned. The execution details will arrive at **EWrapper** (p. **??**):execDetails.

Parameters

reqId	the request's unique identifier.
filter	the filter criteria used to determine which execution reports are returned.

See also

EWrapper::execDetails (p. **??**), **EWrapper::commissionReport** (p. **??**), **ExecutionFilter**

26.9.3.35 reqFamilyCodes()

```
void reqFamilyCodes ( )  [inline]
```

Requests family codes for an account, for instance if it is a FA, IBroker, or associated account.

See also

EWrapper::familyCodes

26.9.3.36 reqFundamentalData()

```
void reqFundamentalData (
            int reqId,
             Contract contract,
            String reportType,
            List< TagValue > fundamentalDataOptions )  [inline]
```

Requests the contract's Reuters or Wall Street Horizons fundamental data. Fundalmental data is returned at **EWrapper::fundamentalData** (p. **??**).

Parameters

reqId	the request's unique identifier.
contract	the contract's description for which the data will be returned.
reportType	there are three available report types: • ReportSnapshot: Company overview • ReportsFinSummary: Financial summary • ReportRatios: Financial ratios • ReportsFinStatements: Financial statements • RESC: Analyst estimates • CalendarReport: Company calendar from Wall Street Horizons

See also

EWrapper::fundamentalData

26.9.3.37 reqGlobalCancel()

```
void reqGlobalCancel ( )  [inline]
```

Cancels all active orders.
This method will cancel ALL open orders including those placed directly from TWS.

See also

cancelOrder

26.9.3.38 reqHeadTimestamp()

```
void reqHeadTimestamp (
            int tickerId,
             Contract contract,
            string whatToShow,
            int useRTH,
            int formatDate )  [inline]
```

Returns the timestamp of earliest available historical data for a contract and data type.

Parameters

tickerId	- an identifier for the request
contract	- contract object for which head timestamp is being requested
whatToShow	- type of data for head timestamp - "BID", "ASK", "TRADES", etc
useRTH	- use regular trading hours only, 1 for yes or 0 for no
formatDate	-
formatDate	set to 1 to obtain the bars' time as yyyyMMdd HH:mm:ss, set to 2 to obtain it like system time format in seconds

See also

headTimeStamp

26.9.3.39 reqHistogramData()

```
void reqHistogramData (
            int tickerId,
             Contract contract,
            bool useRTH,
            string period )  [inline]
```

Returns data histogram of specified contract
.

Parameters

tickerId	- an identifier for the request
contract	- **Contract** object for which histogram is being requested
useRTH	- use regular trading hours only, 1 for yes or 0 for no
period	- period of which data is being requested, e.g. "3 days"

See also

histogramData

26.9.3.40 reqHistoricalData()

```
void reqHistoricalData (
             int tickerId,
              Contract contract,
             string endDateTime,
             string durationString,
             string barSizeSetting,
             string whatToShow,
             int useRTH,
             int formatDate,
             bool keepUpToDate,
             List<  TagValue > chartOptions )  [inline]
```

Requests contracts' historical data. When requesting historical data, a finishing time and date is required along with a duration string. For example, having:

- endDateTime: 20130701 23:59:59 GMT
- durationStr: 3 D will return three days of data counting backwards from July 1st 2013 at 23:59:59 GMT resulting in all the available bars of the last three days until the date and time specified. It is possible to specify a timezone optionally. The resulting bars will be returned in **EWrapper::historicalData**

Parameters

tickerId	the request's unique identifier.
contract	the contract for which we want to retrieve the data.
endDateTime	request's ending time with format yyyyMMdd HH:mm:ss {TMZ}
durationString	the amount of time for which the data needs to be retrieved:

- " S (seconds) - " D (days)
- " W (weeks) - " M (months)
- " Y (years)

Parameters

barSizeSetting	the size of the bar:

- 1 sec
- 5 secs
- 15 secs
- 30 secs
- 1 min

- 2 mins
- 3 mins
- 5 mins
- 15 mins
- 30 mins
- 1 hour
- 1 day

Parameters

whatToShow	the kind of information being retrieved:

- TRADES
- MIDPOINT
- BID
- ASK
- BID_ASK
- HISTORICAL_VOLATILITY
- OPTION_IMPLIED_VOLATILITY
- FEE_RATE
- REBATE_RATE

Parameters

useRTH	set to 0 to obtain the data which was also generated outside of the Regular Trading Hours, set to 1 to obtain only the RTH data
formatDate	set to 1 to obtain the bars' time as yyyyMMdd HH:mm:ss, set to 2 to obtain it like system time format in seconds
keepUpToDate	set to True to received continuous updates on most recent bar data. If True, and endDateTime cannot be specified.

See also

EWrapper::historicalData

26.9.3.41 reqHistoricalNews()

```
void reqHistoricalNews (
            int requestId,
            int conId,
            string providerCodes,
            string startDateTime,
```

```
            string endDateTime,
            int totalResults,
            List<  TagValue > historicalNewsOptions )  [inline]
```

Requests historical news headlines.

Parameters

requestId	
conId	- contract id of ticker
providerCodes	- a '+'-separated list of provider codes
startDateTime	- marks the (exclusive) start of the date range. The format is yyyy-MM-dd HH:mm:ss.0
endDateTime	- marks the (inclusive) end of the date range. The format is yyyy-MM-dd HH:mm:ss.0
totalResults	- the maximum number of headlines to fetch (1 - 300)
historicalNewsOptions	reserved for internal use. Should be defined as null.

See also

EWrapper::historicalNews (p. **??**), **EWrapper::historicalNewsEnd**

26.9.3.42 reqHistoricalTicks()

```
void reqHistoricalTicks (
            int reqId,
             Contract contract,
            string startDateTime,
            string endDateTime,
            int numberOfTicks,
            string whatToShow,
            int useRth,
            bool ignoreSize,
            List<  TagValue > miscOptions )  [inline]
```

Requests historical Time&Sales data for an instrument.

Parameters

reqId	id of the request
contract	**Contract** object that is subject of query
startDate↩ Time,i.e.	"20170701 12:01:00". Uses TWS timezone specified at login.
endDateTime,i.e.	"20170701 13:01:00". In TWS timezone. Exactly one of start time and end time has to be defined.
numberOfTicks	Number of distinct data points. Max currently 1000 per request.
whatToShow	(Bid_Ask, Midpoint, Trades) Type of data requested.
useRth	Data from regular trading hours (1), or all available hours (0)
ignoreSize	A filter only used when the source price is Bid_Ask
miscOptions	should be defined as *null*, reserved for internal use

26.9.3.43 reqIds()

```
void reqIds (
            int numIds )  [inline]
```

Requests the next valid order ID at the current moment.

Parameters

numIds	deprecated- this parameter will not affect the value returned to nextValidId

See also

EWrapper::nextValidId

26.9.3.44 reqManagedAccts()

```
void reqManagedAccts ( )  [inline]
```

Requests the accounts to which the logged user has access to.

See also

EWrapper::managedAccounts

26.9.3.45 reqMarketDataType()

```
void reqMarketDataType (
            int marketDataType )  [inline]
```

Switches data type returned from reqMktData request to "frozen", "delayed" or "delayed-frozen" market data. Requires TWS/IBG v963+.
The API can receive frozen market data from Trader Workstation. Frozen market data is the last data recorded in our system.
During normal trading hours, the API receives real-time market data. Invoking this function with argument 2 requests a switch to frozen data immediately or after the close.
When the market reopens the next data the market data type will automatically switch back to real time if available.

Parameters

marketDataType	by default only real-time (1) market data is enabled sending 1 (real-time) disables frozen, delayed and delayed-frozen market data sending 2 (frozen) enables frozen market data sending 3 (delayed) enables delayed and disables delayed-frozen market data sending 4 (delayed-frozen) enables delayed and delayed-frozen market data

26.9.3.46 reqMarketDepth()

```
void reqMarketDepth (
            int tickerId,
             Contract contract,
            int numRows,
            List< TagValue > mktDepthOptions )  [inline]
```

Requests the contract's market depth (order book).
This request must be direct-routed to an exchange and not smart-routed. The number of simultaneous market depth requests allowed in an account is calculated based on a formula that looks at an accounts equity, commissions, and quote booster packs.

Parameters

tickerId	the request's identifier
contract	the **Contract** for which the depth is being requested
numRows	the number of rows on each side of the order book

See also

cancelMktDepth (p. **??**), **EWrapper::updateMktDepth** (p. **??**), **EWrapper::updateMktDepthL2**

26.9.3.47 reqMarketRule()

```
void reqMarketRule (
            int marketRuleId )  [inline]
```

Requests details about a given market rule
The market rule for an instrument on a particular exchange provides details about how the minimum price increment changes with price
A list of market rule ids can be obtained by invoking reqContractDetails on a particular contract. The returned market rule ID list will provide the market rule ID for the instrument in the correspond valid exchange list in contractDetails.
.

Parameters

marketRuleId	- the id of market rule

See also

EWrapper::marketRule

26.9.3.48 reqMatchingSymbols()

```
void reqMatchingSymbols (
            int reqId,
            string pattern )  [inline]
```

Requests matching stock symbols.

Parameters

reqId	id to specify the request
pattern	- either start of ticker symbol or (for larger strings) company name

See also

EWrapper::symbolSamples

26.9.3.49 reqMktData()

```
void reqMktData (
            int tickerId,
             Contract contract,
            string genericTickList,
            bool snapshot,
            bool regulatorySnaphsot,
            List<  TagValue > mktDataOptions )  [inline]
```

Requests real time market data. Returns market data for an instrument either in real time or 10-15 minutes delayed (depending on the market data type specified)

Parameters

tickerId	the request's identifier
contract	the **Contract** for which the data is being requested

Parameters

genericTickList	comma separated ids of the available generic ticks: • 100 Option Volume (currently for stocks) • 101 Option Open Interest (currently for stocks) • 104 Historical Volatility (currently for stocks) • 105 Average Option Volume (currently for stocks) • 106 Option Implied Volatility (currently for stocks) • 162 Index Future Premium • 165 Miscellaneous Stats • 221 Mark Price (used in TWS P&L computations) • 225 Auction values (volume, price and imbalance) • 233 RTVolume - contains the last trade price, last trade size, last trade time, total volume, VWAP, and single trade flag. • 236 Shortable • 256 Inventory - 258 Fundamental Ratios • 411 Realtime Historical Volatility • 456 IBDividends
snapshot	for users with corresponding real time market data subscriptions. A true value will return a one-time snapshot, while a false value will provide streaming data.
regulatory	snapshot for US stocks requests NBBO snapshots for users which have "US Securities Snapshot Bundle" subscription but not corresponding Network A, B, or C subscription necessary for streaming $*$ market data. One-time snapshot of current market price that will incur a fee of 1 cent to the account per snapshot.

See also

cancelMktData (p. **??**), **EWrapper::tickPrice** (p. **??**), **EWrapper::tickSize** (p. **??**), **EWrapper::tickString** (p. **??**), **EWrapper::tickEFP** (p. **??**), **EWrapper::tickGeneric** (p. **??**), **EWrapper::tickOptionComputation** (p. **??**), **EWrapper::tickSnapshotEnd**

26.9.3.50 reqMktDepthExchanges()

```
void reqMktDepthExchanges ( )  [inline]
```

Requests venues for which market data is returned to updateMktDepthL2 (those with market makers)

See also

EWrapper::mktDepthExchanges

26.9.3.51 reqNewsArticle()

```
void reqNewsArticle (
            int requestId,
            string providerCode,
            string articleId,
            List< TagValue > newsArticleOptions ) [inline]
```

Requests news article body given articleId.

Parameters

requestId	id of the request
providerCode	short code indicating news provider, e.g. FLY
articleId	id of the specific article
newsArticleOptions	reserved for internal use. Should be defined as null.

See also

EWrapper::newsArticle (p. **??**),

26.9.3.52 reqNewsBulletins()

```
void reqNewsBulletins (
            bool allMessages ) [inline]
```

Subscribes to IB's News Bulletins.

Parameters

allMessages	if set to true, will return all the existing bulletins for the current day, set to false to receive only the new bulletins.

See also

cancelNewsBulletin (p. **??**), **EWrapper::updateNewsBulletin**

26.9.3.53 reqNewsProviders()

```
void reqNewsProviders ( ) [inline]
```

Requests news providers which the user has subscribed to.

See also

EWrapper::newsProviders

26.9.3.54 reqOpenOrders()

```
void reqOpenOrders ( )  [inline]
```

Requests all open orders places by this specific API client (identified by the API client id). For client ID 0, this will bind previous manual TWS orders.

See also

reqAllOpenOrders (p. **??**), **reqAutoOpenOrders** (p. **??**), **placeOrder** (p. **??**), **cancelOrder** (p. **??**), **reqGlobalCancel** (p. **??**), **EWrapper::openOrder** (p. **??**), **EWrapper::orderStatus** (p. **??**), **EWrapper::openOrderEnd**

26.9.3.55 reqPnL()

```
void reqPnL (
            int reqId,
            string account,
            string modelCode )  [inline]
```

Creates subscription for real time daily PnL and unrealized PnL updates.

Parameters

account	account for which to receive PnL updates
modelCode	specify to request PnL updates for a specific model

26.9.3.56 reqPnLSingle()

```
void reqPnLSingle (
            int reqId,
            string account,
            string modelCode,
            int conId )  [inline]
```

Requests real time updates for daily PnL of individual positions.

Parameters

reqId	
account	account in which position exists
modelCode	model in which position exists
conId	contract ID (conId) of contract to receive daily PnL updates for. Note: does not return message if invalid conId is entered

26.9.3.57 reqPositions()

```
void reqPositions ( )  [inline]
```

Subscribes to position updates for all accessible accounts. All positions sent initially, and then only updates as positions change.

See also

cancelPositions (p. **??**), **EWrapper::position** (p. **??**), **EWrapper::positionEnd**

26.9.3.58 reqPositionsMulti()

```
void reqPositionsMulti (
            int requestId,
            string account,
            string modelCode )  [inline]
```

Requests position subscription for account and/or model Initially all positions are returned, and then updates are returned for any position changes in real time.

Parameters

requestId	- Request's identifier
account	- If an account Id is provided, only the account's positions belonging to the specified model will be delivered
modelCode	- The code of the model's positions we are interested in.

See also

cancelPositionsMulti (p. **??**), **EWrapper::positionMulti** (p. **??**), **EWrapper::positionMultiEnd**

26.9.3.59 reqRealTimeBars()

```
void reqRealTimeBars (
            int tickerId,
             Contract contract,
            int barSize,
            string whatToShow,
            bool useRTH,
            List<  TagValue > realTimeBarsOptions )  [inline]
```

Requests real time bars
Currently, only 5 seconds bars are provided. This request is subject to the same pacing as any historical data request: no more than 60 API queries in more than 600 seconds.
Real time bars subscriptions are also included in the calculation of the number of Level 1 market data subscriptions allowed in an account.

Parameters

tickerId	the request's unique identifier.
contract	the **Contract** for which the depth is being requested
barSize	currently being ignored
whatToShow	the nature of the data being retrieved: • TRADES • MIDPOINT • BID • ASK
useRTH	set to 0 to obtain the data which was also generated ourside of the Regular Trading Hours, set to 1 to obtain only the RTH data

See also

cancelRealTimeBars (p. **??**), **EWrapper::realtimeBar**

26.9.3.60 reqScannerParameters()

```
void reqScannerParameters ( )  [inline]
```

Requests an XML list of scanner parameters valid in TWS.
Not all parameters are valid from API scanner.

See also

reqScannerSubscription

26.9.3.61 reqScannerSubscription()

```
void reqScannerSubscription (
            int reqId,
             ScannerSubscription subscription,
            List<  TagValue > scannerSubscriptionOptions )  [inline]
```

Starts a subscription to market scan results based on the provided parameters.

Parameters

reqId	the request's identifier
subscription	summary of the scanner subscription including its filters.

See also

reqScannerParameters (p. **??**), **ScannerSubscription** (p. **??**), **EWrapper::scannerData**

26.9.3.62 reqSecDefOptParams()

```
void reqSecDefOptParams (
            int reqId,
            string underlyingSymbol,
            string futFopExchange,
            string underlyingSecType,
            int underlyingConId )  [inline]
```

Requests security definition option parameters for viewing a contract's option chain.

Parameters

reqId	the ID chosen for the request
underlyingSymbol	
futFopExchange	The exchange on which the returned options are trading. Can be set to the empty string "" for all exchanges.
underlyingSecType	The type of the underlying security, i.e. STK
underlyingConId	the contract ID of the underlying security

See also

EWrapper::securityDefinitionOptionParameter

26.9.3.63 reqSmartComponents()

```
void reqSmartComponents (
            int reqId,
            String bboExchange )  [inline]
```

Returns the mapping of single letter codes to exchange names given the mapping identifier.

Parameters

reqId	id of the request
bboExchange	mapping identifier received from **EWrapper.tickReqParams**

See also

EWrapper::smartComponents

26.9.3.64 reqSoftDollarTiers()

```
void reqSoftDollarTiers (
            int reqId )  [inline]
```

Requests pre-defined Soft Dollar Tiers. This is only supported for registered professional advisors and hedge and mutual funds who have configured Soft Dollar Tiers in Account Management. Refer to: https://www.interactivebrokers.com/en/software/am/am/manageaccount/requestsoftdollars.htm?Highlight=soft%20dollar%20tier.

See also

EWrapper::softDollarTiers

26.9.3.65 reqTickByTickData()

```
void reqTickByTickData (
            int requestId,
             Contract contract,
            string tickType,
            int numberOfTicks,
            bool ignoreSize )  [inline]
```

Requests tick-by-tick data.
.

Parameters

reqId	- unique identifier of the request.
contract	- the contract for which tick-by-tick data is requested.
tickType	- tick-by-tick data type: "Last", "AllLast", "BidAsk" or "MidPoint".
numberOfTicks	- number of ticks.
ignoreSize	- ignore size flag.

See also

EWrapper::tickByTickAllLast (p. **??**), **EWrapper::tickByTickBidAsk** (p. **??**), **EWrapper::tickByTickMidPoint** (p. **??**), **Contract**

26.9.3.66 requestFA()

```
void requestFA (
            int faDataType )  [inline]
```

Requests the FA configuration A Financial Advisor can define three different configurations:

1. Groups: offer traders a way to create a group of accounts and apply a single allocation method to all accounts in the group.
2. Profiles: let you allocate shares on an account-by-account basis using a predefined calculation value.
3. Account Aliases: let you easily identify the accounts by meaningful names rather than account numbers. More information at https://www.interactivebrokers.com/en/?f=%2Fen%2Fsoftware%2↩Fpdfhighlights%2FPDF-AdvisorAllocations.php%3Fib_entity%3Dllc

Parameters

faDataType	the configuration to change. Set to 1, 2 or 3 as defined above.

See also

replaceFA

26.9.3.67 subscribeToGroupEvents()

```
void subscribeToGroupEvents (
            int requestId,
            int groupId )  [inline]
```

Integrates API client and TWS window grouping.

Parameters

requestId	is the Id chosen for this subscription request
groupId	is the display group for integration

26.9.3.68 updateDisplayGroup()

```
void updateDisplayGroup (
            int requestId,
            string contractInfo )  [inline]
```

Updates the contract displayed in a TWS Window Group.

Parameters

requestId	is the ID chosen for this request
contractInfo	is an encoded value designating a unique IB contract. Possible values include: 1. none = empty selection 2. contractID - any non-combination contract. Examples 8314 for IBM SMART; 8314 for IBM ARCA 3. combo= if any combo is selected Note: This request from the API does not get a TWS response unless an error occurs.

26.10 EClientErrors Class Reference

Contains all possible errors occurring on the client side. This errors are not sent by the TWS but rather generated as the result of malfunction within the TWS API client.

Static Public Attributes

- static readonly **CodeMsgPair AlreadyConnected** = new **CodeMsgPair**(501, "Already Connected.")
- static readonly **CodeMsgPair CONNECT_FAIL**
- static readonly **CodeMsgPair UPDATE_TWS** = new **CodeMsgPair**(503, "The TWS is out of date and must be upgraded.")
- static readonly **CodeMsgPair NOT_CONNECTED** = new **CodeMsgPair**(504, "Not connected")
- static readonly **CodeMsgPair UNKNOWN_ID** = new **CodeMsgPair**(505, "Fatal Error: Unknown message id.")
- static readonly **CodeMsgPair FAIL_SEND_REQMKT** = new **CodeMsgPair**(510, "Request Market Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANMKT** = new **CodeMsgPair**(511, "Cancel Market Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_ORDER** = new **CodeMsgPair**(512, "Order Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_ACCT** = new **CodeMsgPair**(513, "Account Update Request Sending Error -")
- static readonly **CodeMsgPair FAIL_SEND_EXEC** = new **CodeMsgPair**(514, "Request For Executions Sending Error -")
- static readonly **CodeMsgPair FAIL_SEND_CORDER** = new **CodeMsgPair**(515, "Cancel **Order** Sending Error -")
- static readonly **CodeMsgPair FAIL_SEND_OORDER** = new **CodeMsgPair**(516, "Request Open **Order** Sending Error -")
- static readonly **CodeMsgPair UNKNOWN_CONTRACT** = new **CodeMsgPair**(517, "Unknown contract. Verify the contract details supplied.")
- static readonly **CodeMsgPair FAIL_SEND_REQCONTRACT** = new **CodeMsgPair**(518, "Request **Contract** Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQMKTDEPTH** = new **CodeMsgPair**(519, "Request Market Depth Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANMKTDEPTH** = new **CodeMsgPair**(520, "Cancel Market Depth Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_SERVER_LOG_LEVEL** = new **CodeMsgPair**(521, "Set Server Log Level Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_FA_REQUEST** = new **CodeMsgPair**(522, "FA Information Request Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_FA_REPLACE** = new **CodeMsgPair**(523, "FA Information Replace Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQSCANNER** = new **CodeMsgPair**(524, "Request Scanner Subscription Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANSCANNER** = new **CodeMsgPair**(525, "Cancel Scanner Subscription Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQSCANNERPARAMETERS** = new **CodeMsgPair**(526, "Request Scanner Parameter Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQHISTDATA** = new **CodeMsgPair**(527, "Request Historical Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANHISTDATA** = new **CodeMsgPair**(528, "Request Historical Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQRTBARS** = new **CodeMsgPair**(529, "Request Real-time **Bar** Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANRTBARS** = new **CodeMsgPair**(530, "Cancel Real-time **Bar** Data Sending Error - ")

- static readonly **CodeMsgPair FAIL_SEND_REQCURRTIME** = new **CodeMsgPair**(531, "Request Current Time Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQFUNDDATA** = new **CodeMsgPair**(532, "Request Fundamental Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANFUNDDATA** = new **CodeMsgPair**(533, "Cancel Fundamental Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQCALCIMPLIEDVOLAT** = new **CodeMsgPair**(534, "Request Calculate Implied Volatility Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQCALCOPTIONPRICE** = new **CodeMsgPair**(535, "Request Calculate Option Price Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANCALCIMPLIEDVOLAT** = new **CodeMsgPair**(536, "Cancel Calculate Implied Volatility Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANCALCOPTIONPRICE** = new **CodeMsgPair**(537, "Cancel Calculate Option Price Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQGLOBALCANCEL** = new **CodeMsgPair**(538, "Request Global Cancel Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQMARKETDATATYPE** = new **CodeMsgPair**(539, "Request Market Data Type Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQPOSITIONS** = new **CodeMsgPair**(540, "Request Positions Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANPOSITIONS** = new **CodeMsgPair**(541, "Cancel Positions Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQACCOUNTDATA** = new **CodeMsgPair**(542, "Request Account Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANACCOUNTDATA** = new **CodeMsgPair**(543, "Cancel Account Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_VERIFYREQUEST** = new **CodeMsgPair**(544, "Verify Request Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_VERIFYMESSAGE** = new **CodeMsgPair**(545, "Verify Message Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_QUERYDISPLAYGROUPS** = new **CodeMsgPair**(546, "Query Display Groups Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_SUBSCRIBETOGROUPEVENTS** = new **CodeMsgPair**(547, "Subscribe To Group Events Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_UPDATEDISPLAYGROUP** = new **CodeMsgPair**(548, "Update Display Group Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_UNSUBSCRIBEFROMGROUPEVENTS** = new **CodeMsg↩ Pair**(549, "Unsubscribe From Group Events Sending Error - ")
- static readonly **CodeMsgPair BAD_LENGTH** = new **CodeMsgPair**(507, "Bad message length")
- static readonly **CodeMsgPair BAD_MESSAGE** = new **CodeMsgPair**(508, "Bad message")
- static readonly **CodeMsgPair UNSUPPORTED_VERSION** = new **CodeMsgPair**(506, "Unsupported version")
- static readonly **CodeMsgPair FAIL_SEND_VERIFYANDAUTHREQUEST** = new **CodeMsgPair**(551, "Verify And Auth Request Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_VERIFYANDAUTHMESSAGE** = new **CodeMsgPair**(552, "Verify And Auth Message Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQPOSITIONSMULTI** = new **CodeMsgPair**(553, "Request Positions Multi Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANPOSITIONSMULTI** = new **CodeMsgPair**(554, "Cancel Positions Multi Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQACCOUNTUPDATESMULTI** = new **CodeMsgPair**(555, "Request Account Updates Multi Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANACCOUNTUPDATESMULTI** = new **CodeMsgPair**(556, "Cancel Account Updates Multi Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQSECDEFOPTPARAMS** = new **CodeMsgPair**(557, "Request Security Definition Option Parameters Sending Error - ")

- static readonly **CodeMsgPair FAIL_SEND_REQSOFTDOLLARTIERS** = new **CodeMsgPair**(558, "Request Soft Dollar Tiers Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQFAMILYCODES** = new **CodeMsgPair**(559, "Request Family Codes Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQMATCHINGSYMBOLS** = new **CodeMsgPair**(560, "Request Matching Symbols Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQMKTDEPTHEXCHANGES** = new **CodeMsgPair**(561, "Request Market Depth Exchanges Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQSMARTCOMPONENTS** = new **CodeMsgPair**(562, "Request Smart Components Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQNEWSPROVIDERS** = new **CodeMsgPair**(563, "Request News Providers Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQNEWSARTICLE** = new **CodeMsgPair**(564, "Request News Article Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQHISTORICALNEWS** = new **CodeMsgPair**(565, "Request Historical News Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQHEADTIMESTAMP** = new **CodeMsgPair**(566, "Request Head Time Stamp Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQHISTOGRAMDATA** = new **CodeMsgPair**(567, "Request Histogram Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANCELHISTOGRAMDATA** = new **CodeMsgPair**(568, "Cancel Request Histogram Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANCELHEADTIMESTAMP** = new **CodeMsgPair**(569, "Cancel Head Time Stamp Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQMARKETRULE** = new **CodeMsgPair**(570, "Request Market Rule Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQPNL** = new **CodeMsgPair**(571, "Request PnL Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANCELPNL** = new **CodeMsgPair**(572, "Cancel PnL Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQPNLSINGLE** = new **CodeMsgPair**(573, "Request PnL Single Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANCELPNLSINGLE** = new **CodeMsgPair**(574, "Cancel PnL Single Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQHISTORICALTICKS** = new **CodeMsgPair**(575, "Request Historical Ticks Error - ")
- static readonly **CodeMsgPair FAIL_SEND_REQTICKBYTICKDATA** = new **CodeMsgPair**(576, "Request Tick-By-Tick Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_SEND_CANCELTICKBYTICKDATA** = new **CodeMsgPair**(577, "Cancel Tick-By-Tick Data Sending Error - ")
- static readonly **CodeMsgPair FAIL_GENERIC** = new **CodeMsgPair**(-1, "Specific error message needs to be given for these requests! ")

26.10.1 Detailed Description

Contains all possible errors occurring on the client side. This errors are not sent by the TWS but rather generated as the result of malfunction within the TWS API client.

26.10.2 Member Data Documentation

26.10.2.1 CONNECT_FAIL

```
readonly  CodeMsgPair CONNECT_FAIL  [static]
```

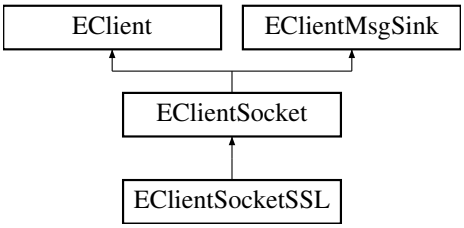
Initial value:

```
= new CodeMsgPair(502, @"Couldn't connect to TWS. Confirm that ""Enable ActiveX and Socket Clients""
                          is enabled and connection port is the same as ""Socket Port"" on the TWS ""
      Edit->Global Configuration...->API->Settings"" menu.
                          Live Trading ports: TWS: 7496; IB Gateway: 4001. Simulated Trading ports for
       new installations of version 954.1 or newer:
                          TWS: 7497; IB Gateway: 4002")
```

26.11 EClientSocket Class Reference

TWS/Gateway client class This client class contains all the available methods to communicate with IB. Up to 32 clients can be connected to a single instance of the TWS/Gateway simultaneously. From herein, the TWS/Gateway will be referred to as the Host.

Inheritance diagram for EClientSocket:



Public Member Functions

- **EClientSocket** (**EWrapper** wrapper, **EReaderSignal** eReaderSignal)
- void **eConnect** (string host, int port, int clientId)
- void **eConnect** (string host, int port, int clientId, bool extraAuth)

  Creates socket connection to TWS/IBG.
- void **redirect** (string host)

  Redirects connection to different host.
- override void **eDisconnect** (bool resetState=true)

  Closes the socket connection and terminates its thread.

Protected Member Functions

- virtual Stream **createClientStream** (string host, int port)
- override uint **prepareBuffer** (BinaryWriter paramsList)
- override void **CloseAndSend** (BinaryWriter request, uint lengthPos)

Additional Inherited Members

26.11.1 Detailed Description

TWS/Gateway client class This client class contains all the available methods to communicate with IB. Up to 32 clients can be connected to a single instance of the TWS/Gateway simultaneously. From herein, the TWS/Gateway will be referred to as the Host.

26.11.2 Member Function Documentation

26.11.2.1 eConnect()

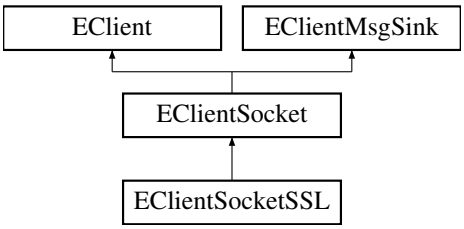
```
void eConnect (
             string host,
             int port,
             int clientId )  [inline]
```

Creates socket connection to TWS/IBG. This earlier version of eConnect does not have extraAuth parameter.

26.12 EClientSocketSSL Class Reference

Implements a Secure Socket Layer (SSL) on top of the **EClientSocket** class.

Inheritance diagram for EClientSocketSSL:



Public Member Functions

- **EClientSocketSSL** (**EWrapper** wrapper, **EReaderSignal** signal)

Protected Member Functions

- override Stream **createClientStream** (string host, int port)

Additional Inherited Members

26.12.1 Detailed Description

Implements a Secure Socket Layer (SSL) on top of the **EClientSocket** class.

26.13 EReader Class Reference

Captures incoming messages to the API client and places them into a queue.

Public Member Functions

- **EReader** (**EClientSocket** clientSocket, **EReaderSignal** signal)
- void **Start** ()
- void **processMsgs** ()
- bool **putMessageToQueue** ()

26.13.1 Detailed Description

Captures incoming messages to the API client and places them into a queue.

26.14 EReaderSignal Interface Reference

Notifies the thread reading information from the TWS whenever there are messages ready to be consumed. Not currently used in Python API.

Public Member Functions

- void **issueSignal** ()

  Issues a signal to the consuming thread when there are things to be consumed.
- void **waitForSignal** ()

  Makes the consuming thread waiting until a signal is issued.

26.14.1 Detailed Description

Notifies the thread reading information from the TWS whenever there are messages ready to be consumed. Not currently used in Python API.

26.15 EWrapper Interface Reference

This interface's methods are used by the TWS/Gateway to communicate with the API client. Every API client application needs to implement this interface in order to handle all the events generated by the TWS/Gateway. Almost every **EClientSocket** method call will result in at least one event delivered here.

Public Member Functions

- void **error** (Exception e)

  *Handles errors generated within the API itself. If an exception is thrown within the API code it will be notified here. Possible cases include errors while reading the information from the socket or even mishandling at **EWrapper** (p. **??**)'s implementing class.*
- void **error** (string str)
- void **error** (int id, int errorCode, string errorMsg)

  Errors sent by the TWS are received here.
- void **currentTime** (long time)

  TWS's current time. TWS is synchronized with the server (not local computer) using NTP and this function will receive the current time in TWS.
- void **tickPrice** (int tickerId, int field, double price, **TickAttrib** attribs)

  Market data tick price callback. Handles all price related ticks. Every tickPrice callback is followed by a tickSize. A tickPrice value of -1 or 0 followed by a tickSize of 0 indicates there is no data for this field currently available, whereas a tickPrice with a positive tickSize indicates an active quote of 0 (typically for a combo contract).
- void **tickSize** (int tickerId, int field, int size)

  Market data tick size callback. Handles all size-related ticks.
- void **tickString** (int tickerId, int field, string value)

  Market data callback. Every tickPrice is followed by a tickSize. There are also independent tickSize callbacks anytime the tickSize changes, and so there will be duplicate tickSize messages following a tickPrice.
- void **tickGeneric** (int tickerId, int field, double value)

  Market data callback.
- void **tickEFP** (int tickerId, int tickType, double basisPoints, string formattedBasisPoints, double impliedFuture, int holdDays, string futureLastTradeDate, double dividendImpact, double dividendsToLastTradeDate)

  Exchange for Physicals.
- void **deltaNeutralValidation** (int reqId, **DeltaNeutralContract** deltaNeutralContract)
  - *Upon accepting a Delta-Neutral DN RFQ(request for quote), the server sends a **deltaNeutralValidation()** message with the **DeltaNeutralContract** structure. If the delta and price fields are empty in the original request, the confirmation will contain the current values from the server. These values are locked when RFQ is processed and remain locked unitl the RFQ is cancelled.*
- void **tickOptionComputation** (int tickerId, int field, double impliedVolatility, double delta, double optPrice, double pvDividend, double gamma, double vega, double theta, double undPrice)

  Receive's option specific market data. This method is called when the market in an option or its underlier moves. TWS's option model volatilities, prices, and deltas, along with the present value of dividends expected on that options underlier are received.
- void **tickSnapshotEnd** (int tickerId)

  When requesting market data snapshots, this market will indicate the snapshot reception is finished. Expected to occur 11 seconds after beginning of request.
- void **nextValidId** (int orderId)

  *Receives next valid order id. Will be invoked automatically upon successfull API client connection, or after call to **EClient::reqIds** Important: the next valid order ID is only valid at the time it is received.*
- void **managedAccounts** (string accountsList)

  Receives a comma-separated string with the managed account ids. Occurs automatically on initial API client connection.
- void **connectionClosed** ()

  Callback to indicate the API connection has closed. Following a API <-> TWS broken socket connection, this function is not called automatically but must be triggered by API client code.
- void **accountSummary** (int reqId, string account, string tag, string value, string currency)

  Receives the account information. This method will receive the account information just as it appears in the TWS' Account Summary Window.
- void **accountSummaryEnd** (int reqId)

  notifies when all the accounts' information has ben received. Requires TWS 967+ to receive accountSummaryEnd in linked account structures.
- void **bondContractDetails** (int reqId, **ContractDetails** contract)

Delivers the Bond contract data after this has been requested via reqContractDetails.

- void **updateAccountValue** (string key, string value, string currency, string accountName)

  Receives the subscribed account's information. Only one account can be subscribed at a time. After the initial callback to updateAccountValue, callbacks only occur for values which have changed. This occurs at the time of a position change, or every 3 minutes at most. This frequency cannot be adjusted.
- void **updatePortfolio** (**Contract** contract, double **position**, double marketPrice, double marketValue, double averageCost, double unrealizedPNL, double realizedPNL, string accountName)

  Receives the subscribed account's portfolio. This function will receive only the portfolio of the subscribed account. If the portfolios of all managed accounts are needed, refer to EClientSocket::reqPosition After the initial callback to updatePortfolio, callbacks only occur for positions which have changed.
- void **updateAccountTime** (string timestamp)

  Receives the last time on which the account was updated.
- void **accountDownloadEnd** (string account)

  Notifies when all the account's information has finished.
- void **orderStatus** (int orderId, string status, double filled, double remaining, double avgFillPrice, int permId, int parentId, double lastFillPrice, int clientId, string whyHeld, double mktCapPrice)

  Gives the up-to-date information of an order every time it changes. Often there are duplicate orderStatus messages.
- void **openOrder** (int orderId, **Contract** contract, **Order** order, **OrderState** orderState)

  Feeds in currently open orders.
- void **openOrderEnd** ()

  Notifies the end of the open orders' reception.
- void **contractDetails** (int reqId, **ContractDetails** contractDetails)

  *receives the full contract's definitons This method will return all contracts matching the requested via **EClientSocket::reqContractDetails** (p. **??**). For example, one can obtain the whole option chain with it.*
- void **contractDetailsEnd** (int reqId)

  After all contracts matching the request were returned, this method will mark the end of their reception.
- void **execDetails** (int reqId, **Contract** contract, **Execution** execution)

  Provides the executions which happened in the last 24 hours.
- void **execDetailsEnd** (int reqId)

  *indicates the end of the **Execution** reception.*
- void **commissionReport** (**CommissionReport** commissionReport)

  *provides the **CommissionReport** of an **Execution***
- void **fundamentalData** (int reqId, string data)

  returns Reuters' Fundamental data
- void **historicalData** (int reqId, **Bar** bar)

  returns the requested historical data bars
- void **historicalDataUpdate** (int reqId, **Bar** bar)

  Receives bars in real time if keepUpToDate is set as True in reqHistoricalData. Similar to realTimeBars function, except returned data is a composite of historical data and real time data that is equivalent to TWS chart functionality to keep charts up to date. Returned bars are successfully updated using real time data.
- void **historicalDataEnd** (int reqId, string start, string end)

  Marks the ending of the historical bars reception.
- void **marketDataType** (int reqId, int marketDataType)

  *Returns the market data type (real-time, frozen, delayed, delayed-frozen) of ticker sent by **EClientSocket::reqMktData** when TWS switches from real-time to frozen and back and from delayed to delayed-frozen and back.*
- void **updateMktDepth** (int tickerId, int **position**, int operation, int side, double price, int size)

  Returns the order book.
- void **updateMktDepthL2** (int tickerId, int **position**, string marketMaker, int operation, int side, double price, int size)

  Returns the order book.
- void **updateNewsBulletin** (int msgId, int msgType, String message, String origExchange)

  provides IB's bulletins
- void **position** (string account, **Contract** contract, double pos, double avgCost)

provides the portfolio's open positions.

- void **positionEnd** ()

  Indicates all the positions have been transmitted.

- void **realtimeBar** (int reqId, long time, double open, double high, double low, double close, long volume, double WAP, int count)

  updates the real time 5 seconds bars

- void **scannerParameters** (string xml)

  provides the xml-formatted parameters available from TWS market scanners (not all available in API).

- void **scannerData** (int reqId, int rank, **ContractDetails** **contractDetails**, string distance, string benchmark, string projection, string legsStr)

  provides the data resulting from the market scanner request.

- void **scannerDataEnd** (int reqId)

  Indicates the scanner data reception has terminated.

- void **receiveFA** (int faDataType, string faXmlData)

  receives the Financial Advisor's configuration available in the TWS

- void **verifyMessageAPI** (string apiData)

  Not generally available.

- void **verifyCompleted** (bool isSuccessful, string errorText)

  Not generally available.

- void **verifyAndAuthMessageAPI** (string apiData, string xyzChallenge)

  Not generally available.

- void **verifyAndAuthCompleted** (bool isSuccessful, string errorText)

  Not generally available.

- void **displayGroupList** (int reqId, string groups)

  a one-time response to querying the display groups.

- void **displayGroupUpdated** (int reqId, string contractInfo)

  call triggered once after receiving the subscription request, and will be sent again if the selected contract in the subscribed $*$ display group has changed.

- void **connectAck** ()

  callback initially acknowledging connection attempt connection handshake not complete until nextValidID is received

- void **positionMulti** (int requestId, string account, string modelCode, **Contract** contract, double pos, double avgCost)

  provides the portfolio's open positions.

- void **positionMultiEnd** (int requestId)

  Indicates all the positions have been transmitted.

- void **accountUpdateMulti** (int requestId, string account, string modelCode, string key, string value, string currency)

  provides the account updates.

- void **accountUpdateMultiEnd** (int requestId)

  Indicates all the account updates have been transmitted.

- void **securityDefinitionOptionParameter** (int reqId, string exchange, int underlyingConId, string trading↩ Class, string multiplier, HashSet< string > expirations, HashSet< double > strikes)

  returns the option chain for an underlying on an exchange specified in reqSecDefOptParams There will be multiple callbacks to securityDefinitionOptionParameter if multiple exchanges are specified in reqSecDefOptParams

- void **securityDefinitionOptionParameterEnd** (int reqId)

  called when all callbacks to securityDefinitionOptionParameter are complete

- void **softDollarTiers** (int reqId, **SoftDollarTier**[ ] tiers)

  called when receives Soft Dollar Tier configuration information

- void **familyCodes** (FamilyCode[ ] familyCodes)

  returns array of family codes

- void **symbolSamples** (int reqId, ContractDescription[ ] contractDescriptions)

  returns array of sample contract descriptions

- void **mktDepthExchanges** (DepthMktDataDescription[ ] depthMktDataDescriptions)

  called when receives Depth Market Data Descriptions
- void **tickNews** (int tickerId, long timeStamp, string providerCode, string articleId, string headline, string extraData)

  ticks with news headline
- void **smartComponents** (int reqId, Dictionary< int, KeyValuePair< string, char >> theMap)

  bit number to exchange + exchange abbreviation dictionary
- void **tickReqParams** (int tickerId, double minTick, string bboExchange, int snapshotPermissions)

  tick with BOO exchange and snapshot permissions
- void **newsProviders** (NewsProvider[ ] newsProviders)

  returns array of subscribed API news providers for this user
- void **newsArticle** (int requestId, int articleType, string articleText)

  called when receives News Article
- void **historicalNews** (int requestId, string time, string providerCode, string articleId, string headline)

  returns news headline
- void **historicalNewsEnd** (int requestId, bool hasMore)

  returns news headlines end marker
- void **headTimestamp** (int reqId, string headTimestamp)
  - *returns beginning of data for contract for specified data type*
- void **histogramData** (int reqId, HistogramEntry[ ] data)

  returns data histogram
- void **rerouteMktDataReq** (int reqId, int conId, string exchange)
  - *returns conId and exchange for CFD market data request re-route*
- void **rerouteMktDepthReq** (int reqId, int conId, string exchange)

  returns the conId and exchange for an underlying contract when a request is made for level 2 data for an instrument which does not have data in IB's database. For example stock CFDs and index CFDs.

- void **marketRule** (int marketRuleId, PriceIncrement[ ] priceIncrements)

  returns minimum price increment structure for a particular market rule ID market rule IDs for an instrument on valid exchanges can be obtained from the contractDetails object for that contract
- void **pnl** (int reqId, double dailyPnL, double unrealizedPnL, double realizedPnL)

  receives PnL updates in real time for the daily PnL and the total unrealized PnL for an account
- void **pnlSingle** (int reqId, int pos, double dailyPnL, double unrealizedPnL, double realizedPnL, double value)

  receives real time updates for single position daily PnL values
- void **historicalTicks** (int reqId, **HistoricalTick**[ ] ticks, bool done)
- void **historicalTicksBidAsk** (int reqId, **HistoricalTickBidAsk**[ ] ticks, bool done)
- void **historicalTicksLast** (int reqId, **HistoricalTickLast**[ ] ticks, bool done)
- void **tickByTickAllLast** (int reqId, int tickType, long time, double price, int size, **TickAttrib** attribs, string exchange, string specialConditions)

  returns "Last" or "AllLast" tick-by-tick real-time tick
- void **tickByTickBidAsk** (int reqId, long time, double bidPrice, double askPrice, int bidSize, int askSize, **TickAttrib** attribs)

  returns "BidAsk" tick-by-tick real-time tick
- void **tickByTickMidPoint** (int reqId, long time, double midPoint)

  returns "MidPoint" tick-by-tick real-time tick

26.15.1 Detailed Description

This interface's methods are used by the TWS/Gateway to communicate with the API client. Every API client application needs to implement this interface in order to handle all the events generated by the TWS/Gateway. Almost every **EClientSocket** method call will result in at least one event delivered here.

See also

EClientSocket class

26.15.2 Member Function Documentation

26.15.2.1 accountDownloadEnd()

```
void accountDownloadEnd (
            string account )
```

Notifies when all the account's information has finished.

Parameters

account	the account's id

See also

updateAccountTime (p. **??**), **updatePortfolio** (p. **??**), **updateAccountValue** (p. **??**), **EClientSocket::req↩ AccountUpdates**

26.15.2.2 accountSummary()

```
void accountSummary (
            int reqId,
            string account,
            string tag,
            string value,
            string currency )
```

Receives the account information. This method will receive the account information just as it appears in the TWS' Account Summary Window.

Parameters

reqId	the request's unique identifier.
account	the account id

Parameters

tag	the account's attribute being received. • AccountType — Identifies the IB account structure • NetLiquidation — The basis for determining the price of the assets in your account. Total cash value + stock value + options value + bond value • TotalCashValue — Total cash balance recognized at the time of trade + futures PNL • SettledCash — Cash recognized at the time of settlement - purchases at the time of trade - commissions - taxes - fees • AccruedCash — Total accrued cash value of stock, commodities and securities • BuyingPower — Buying power serves as a measurement of the dollar value of securities that one may purchase in a securities account without depositing additional funds • EquityWithLoanValue — Forms the basis for determining whether a client has the necessary assets to either initiate or maintain security positions. Cash + stocks + bonds + mutual funds • PreviousEquityWithLoanValue — Marginable Equity with Loan value as of 16:00 ET the previous day • GrossPositionValue — The sum of the absolute value of all stock and equity option positions • RegTEquity — Regulation T equity for universal account • RegTMargin — Regulation T margin for universal account • SMA — Special Memorandum Account: Line of credit created when the market value of securities in a Regulation T account increase in value • InitMarginReq — Initial Margin requirement of whole portfolio • MaintMarginReq — Maintenance Margin requirement of whole portfolio • AvailableFunds — This value tells what you have available for trading • ExcessLiquidity — This value shows your margin cushion, before liquidation • Cushion — Excess liquidity as a percentage of net liquidation value • FullInitMarginReq — Initial Margin of whole portfolio with no discounts or intraday credits • FullMaintMarginReq — Maintenance Margin of whole portfolio with no discounts or intraday credits • FullAvailableFunds — Available funds of whole portfolio with no discounts or intraday credits • FullExcessLiquidity — Excess liquidity of whole portfolio with no discounts or intraday credits • LookAheadNextChange — Time when look-ahead values take effect • LookAheadInitMarginReq — Initial Margin requirement of whole portfolio as of next period's margin change • LookAheadMaintMarginReq — Maintenance Margin requirement of whole portfolio as of next period's margin change • LookAheadAvailableFunds — This value reflects your available funds at the next margin change • LookAheadExcessLiquidity — This value reflects your excess liquidity at the next margin change • HighestSeverity — A measure of how close the account is to liquidation • DayTradesRemaining — The Number of Open/Close trades a user could put on before Pattern Day Trading is detected. A value of "-1" means that the user can put on unlimited day trades. • Leverage — GrossPositionValue / NetLiquidation

Parameters

value	the account's attribute's value.
currency	the currency on which the value is expressed.

See also

accountSummaryEnd (p. **??**), **EClientSocket::reqAccountSummary**

26.15.2.3 accountSummaryEnd()

```
void accountSummaryEnd (
             int reqId )
```

notifies when all the accounts' information has ben received. Requires TWS 967+ to receive accountSummaryEnd in linked account structures.

Parameters

reqId	the request's identifier.

See also

accountSummary (p. **??**), **EClientSocket::reqAccountSummary**

26.15.2.4 accountUpdateMulti()

```
void accountUpdateMulti (
             int requestId,
             string account,
             string modelCode,
             string key,
             string value,
             string currency )
```

provides the account updates.

Parameters

requestId	the id of request
account	the account with updates
modelCode	the model code with updates
key	the name of parameter
value	the value of parameter
currency	the currency of parameter

See also

accountUpdateMultiEnd (p. **??**), **EClient::reqAccountUpdatesMulti**

26.15.2.5 accountUpdateMultiEnd()

```
void accountUpdateMultiEnd (
            int requestId )
```

Indicates all the account updates have been transmitted.

See also

EWrapper::accountUpdateMulti (p. **??**), **EClientSocket::reqAccountUpdatesMulti**

26.15.2.6 bondContractDetails()

```
void bondContractDetails (
            int reqId,
             ContractDetails contract )
```

Delivers the Bond contract data after this has been requested via reqContractDetails.

Parameters

reqId	the request's identifier
contract	the bond contract's information.

See also

reqContractDetails

26.15.2.7 commissionReport()

```
void commissionReport (
            CommissionReport commissionReport )
```

provides the **CommissionReport** of an **Execution**

See also

execDetails (p. **??**), **execDetailsEnd** (p. **??**), **EClientSocket::reqExecutions** (p. **??**), **CommissionReport**

26.15.2.8 connectionClosed()

```
void connectionClosed ( )
```

Callback to indicate the API connection has closed. Following a API <-> TWS broken socket connection, this function is not called automatically but must be triggered by API client code.

See also

EClientSocket::eDisconnect

26.15.2.9 contractDetails()

```
void contractDetails (
            int reqId,
             ContractDetails contractDetails )
```

receives the full contract's definitons This method will return all contracts matching the requested via **EClient↩ Socket::reqContractDetails** (p. **??**). For example, one can obtain the whole option chain with it.

Parameters

reqId	the unique request identifier
contractDetails	the instrument's complete definition.

See also

contractDetailsEnd (p. **??**), **EClientSocket::reqContractDetails**

26.15.2.10 contractDetailsEnd()

```
void contractDetailsEnd (
            int reqId )
```

After all contracts matching the request were returned, this method will mark the end of their reception.

Parameters

req↩ Id	the request's identifier

See also

contractDetails (p. **??**), **EClientSocket::reqContractDetails**

26.15.2.11 currentTime()

```
void currentTime (
            long time )
```

TWS's current time. TWS is synchronized with the server (not local computer) using NTP and this function will receive the current time in TWS.

See also

EClient::reqCurrentTime

26.15.2.12 deltaNeutralValidation()

```
void deltaNeutralValidation (
            int reqId,
             DeltaNeutralContract deltaNeutralContract )
```

- Upon accepting a Delta-Neutral DN RFQ(request for quote), the server sends a **deltaNeutralValidation()** message with the **DeltaNeutralContract** structure. If the delta and price fields are empty in the original request, the confirmation will contain the current values from the server. These values are locked when RFQ is processed and remain locked unitl the RFQ is cancelled.

Parameters

reqId	the request's identifier.
deltaNeutralContract	Delta-Neutral **Contract**

26.15.2.13 displayGroupList()

```
void displayGroupList (
            int reqId,
            string groups )
```

a one-time response to querying the display groups.

Parameters

reqId	the ID of the request
lt	returns a list of integers representing visible Group ID separated by the "|" character, and sorted by most used group first.

See also

EClientSocket::queryDisplayGroups

26.15.2.14 displayGroupUpdated()

```
void displayGroupUpdated (
            int reqId,
            string contractInfo )
```

call triggered once after receiving the subscription request, and will be sent again if the selected contract in the subscribed ∗ display group has changed.

Parameters

reqId	the ID of the request
contractInfo	

See also

EClient::subscribeToGroupEvents

26.15.2.15 error() [1/3]

```
void error (
            Exception e )
```

Handles errors generated within the API itself. If an exception is thrown within the API code it will be notified here. Possible cases include errors while reading the information from the socket or even mishandling at **EWrapper** (p. **??**)'s implementing class.

Parameters

e	the thrown exception.

26.15.2.16 error() [2/3]

```
void error (
            string str )
```

Parameters

str	The error message received.

26.15.2.17 error() [3/3]

```
void error (
            int id,
            int errorCode,
            string errorMsg )
```

Errors sent by the TWS are received here.

Parameters

id	the request identifier which generated the error. Note: -1 will indicate a notification and not true error condition.
errorCode	the code identifying the error.
errorMsg	error's description. Currently Latin-1 encoded error messages are supported. If logged into TWS in a different language it is recommended to enabled the setting in TWS Global Configuration -> API -> Settings -> Show API errors in English.

26.15.2.18 execDetails()

```
void execDetails (
            int reqId,
             Contract contract,
             Execution execution )
```

Provides the executions which happened in the last 24 hours.

Parameters

reqId	the request's identifier
contract	the **Contract** of the **Order**
execution	the **Execution** details.

See also

execDetailsEnd (p. **??**), **commissionReport** (p. **??**), **EClientSocket::reqExecutions** (p. **??**), **Execution**

26.15.2.19 execDetailsEnd()

```
void execDetailsEnd (
            int reqId )
```

indicates the end of the **Execution** reception.

Parameters

reqId	the request's identifier

See also

execDetails (p. **??**), **commissionReport** (p. **??**), **EClientSocket::reqExecutions**

26.15.2.20 familyCodes()

```
void familyCodes (
            FamilyCode [] familyCodes )
```

returns array of family codes

Parameters

FamilyCode[]	

See also

EClient::reqFamilyCodes

26.15.2.21 fundamentalData()

```
void fundamentalData (
            int reqId,
            string data )
```

returns Reuters' Fundamental data

Parameters

reqId	the request's identifier
data	Reuthers xml-formatted fundamental data

See also

EClientSocket::reqFundamentalData

26.15.2.22 headTimestamp()

```
void headTimestamp (
            int reqId,
            string headTimestamp )
```

- returns beginning of data for contract for specified data type

Parameters

requestId	
headTimestamp	- string identifying earliest data date

See also

EClient::reqHeadTimestamp

26.15.2.23 histogramData()

```
void histogramData (
            int reqId,
            HistogramEntry [] data )
```

returns data histogram

Parameters

request↩ Id	
data	- returned Tuple of histogram data, number of trades at specified price level

See also

EClient::reqHistogramData

26.15.2.24 historicalData()

```
void historicalData (
            int reqId,
             Bar bar )
```

returns the requested historical data bars

Parameters

reqId	the request's identifier
bar	the OHLC historical data **Bar** (p. **??**). The time zone of the bar is the time zone chosen on the TWS login screen. Smallest bar size is 1 second.

See also

EClientSocket::reqHistoricalData

26.15.2.25 historicalDataUpdate()

```
void historicalDataUpdate (
            int reqId,
             Bar bar )
```

Receives bars in real time if keepUpToDate is set as True in reqHistoricalData. Similar to realTimeBars function, except returned data is a composite of historical data and real time data that is equivalent to TWS chart functionality to keep charts up to date. Returned bars are successfully updated using real time data.

Parameters

reqId	the requests identifier
bar	the OHLC historical data **Bar** (p. **??**). The time zone of the bar is the time zone chosen on the TWS login screen. Smallest bar size is 1 second.

26.15.2.26 historicalNews()

```
void historicalNews (
            int requestId,
            string time,
            string providerCode,
            string articleId,
            string headline )
```

returns news headline

Parameters

requestId	
time	
providerCode	
articleId	
headline	

See also

EClient::reqHistoricalNews

26.15.2.27 historicalNewsEnd()

```
void historicalNewsEnd (
            int requestId,
            bool hasMore )
```

returns news headlines end marker

Parameters

requestId	
hasMore	- indicates whether there are more results available, false otherwise

See also

EClient::reqHistoricalNews

26.15.2.28 historicalTicks()

```
void historicalTicks (
            int reqId,
             HistoricalTick [] ticks,
            bool done )
```

Parameters

reqId	
ticks	list of **HistoricalTick** data
done	flag to indicate if all historical tick data has been received

26.15.2.29 historicalTicksBidAsk()

```
void historicalTicksBidAsk (
            int reqId,
             HistoricalTickBidAsk [] ticks,
            bool done )
```

Parameters

reqId	
ticks	list of HistoricalBidAsk data
done	flag to indicate if all historical tick data has been received

26.15.2.30 historicalTicksLast()

```
void historicalTicksLast (
            int reqId,
             HistoricalTickLast [] ticks,
            bool done )
```

Parameters

reqId	
ticks	list of **HistoricalTickLast** data
done	flag to indicate if all historical tick data has been received

26.15.2.31 managedAccounts()

```
void managedAccounts (
            string accountsList )
```

Receives a comma-separated string with the managed account ids. Occurs automatically on initial API client connection.

See also

EClientSocket::reqManagedAccts

26.15.2.32 marketDataType()

```
void marketDataType (
            int reqId,
            int marketDataType )
```

Returns the market data type (real-time, frozen, delayed, delayed-frozen) of ticker sent by **EClientSocket::reqMktData** when TWS switches from real-time to frozen and back and from delayed to delayed-frozen and back.

Parameters

reqId	the id of ticker sent in reqMktData
marketDataType	means that now API starts to tick with the folloiwng market data: 1 for real-time, 2 for frozen, 3 for delayed, 4 for delayed-frozen

See also

EClientSocket::reqMarketDataType (p. **??**), **EClientSocket::reqMktData**

26.15.2.33 marketRule()

```
void marketRule (
            int marketRuleId,
            PriceIncrement [] priceIncrements )
```

returns minimum price increment structure for a particular market rule ID market rule IDs for an instrument on valid exchanges can be obtained from the contractDetails object for that contract

Parameters

marketRuleId	
PriceIncrement[]	

See also

EClient::reqMarketRule

26.15.2.34 mktDepthExchanges()

```
void mktDepthExchanges (
            DepthMktDataDescription [] depthMktDataDescriptions )
```

called when receives Depth Market Data Descriptions

Parameters

descriptions	Stores a list of DepthMktDataDescriprion

See also

EClient::reqMktDepthExchanges

26.15.2.35 newsArticle()

```
void newsArticle (
            int requestId,
            int articleType,
            string articleText )
```

called when receives News Article

Parameters

requestId	The request ID used in the call to **EClient::reqNewsArticle**
articleType	The type of news article (0 - plain text or html, 1 - binary data / pdf)
articleText	The body of article (if articleType == 1, the binary data is encoded using the Base64 scheme)

See also

EClient::reqNewsArticle

26.15.2.36 newsProviders()

```
void newsProviders (
            NewsProvider [] newsProviders )
```

returns array of subscribed API news providers for this user

Parameters

NewsProvider[]	

See also

EClient::reqNewsProviders

26.15.2.37 nextValidId()

```
void nextValidId (
            int orderId )
```

Receives next valid order id. Will be invoked automatically upon successfull API client connection, or after call to **EClient::reqIds** Important: the next valid order ID is only valid at the time it is received.

Parameters

orderId	the next order id

See also

EClientSocket::reqIds

26.15.2.38 openOrder()

```
void openOrder (
            int orderId,
             Contract contract,
             Order order,
             OrderState orderState )
```

Feeds in currently open orders.

Parameters

orderId	the order's unique id
contract	the order's **Contract** (p. **??**).
order	the currently active **Order** (p. **??**).
orderState	the order's **OrderState**

See also

orderStatus (p. **??**), **openOrderEnd** (p. **??**), **EClientSocket::placeOrder** (p. **??**), **EClientSocket::reqAll↩ OpenOrders** (p. **??**), **EClientSocket::reqAutoOpenOrders**

26.15.2.39 openOrderEnd()

```
void openOrderEnd ( )
```

Notifies the end of the open orders' reception.

See also

orderStatus (p. **??**), **openOrder** (p. **??**), **EClientSocket::placeOrder** (p. **??**), **EClientSocket::reqAllOpen↩ Orders** (p. **??**), **EClientSocket::reqAutoOpenOrders**

26.15.2.40 orderStatus()

```
void orderStatus (
            int orderId,
            string status,
            double filled,
            double remaining,
            double avgFillPrice,
            int permId,
            int parentId,
            double lastFillPrice,
            int clientId,
            string whyHeld,
            double mktCapPrice )
```

Gives the up-to-date information of an order every time it changes. Often there are duplicate orderStatus messages.

Parameters

orderId	the order's client id.
status	the current status of the order. Possible values: PendingSubmit - indicates that you have transmitted the order, but have not yet received confirmation that it has been accepted by the order destination. PendingCancel - indicates that you have sent a request to cancel the order but have not yet received cancel confirmation from the order destination. At this point, your order is not confirmed canceled. It is not guaranteed that the cancellation will be successful. PreSubmitted - indicates that a simulated order type has been accepted by the IB system and that this order has yet to be elected. The order is held in the IB system until the election criteria are met. At that time the order is transmitted to the order destination as specified . Submitted - indicates that your order has been accepted by the system. ApiCancelled - after an order has been submitted and before it has been acknowledged, an API client client can request its cancelation, producing this state. Cancelled - indicates that the balance of your order has been confirmed canceled by the IB system. This could occur unexpectedly when IB or the destination has rejected your order. Filled - indicates that the order has been completely filled. Market orders executions will not always trigger a Filled status. Inactive - indicates that the order was received by the system but is no longer active because it was rejected or canceled.
filled	number of filled positions.
remaining	the remnant positions.
avgFillPrice	average filling price.
permId	the order's permId used by the TWS to identify orders.
parentId	parent's id. Used for bracket and auto trailing stop orders.
lastFillPrice	price at which the last positions were filled.
clientId	API client which submitted the order.
whyHeld	this field is used to identify an order held when TWS is trying to locate shares for a short sell. The value used to indicate this is 'locate'.
mktCapPrice	If an order has been capped, this indicates the current capped price. Requires TWS 967+ and API v973.04+. Python API specifically requires API v973.06+.

See also

openOrder (p. **??**), **openOrderEnd** (p. **??**), **EClientSocket::placeOrder** (p. **??**), **EClientSocket::reqAll↩ OpenOrders** (p. **??**), **EClientSocket::reqAutoOpenOrders**

26.15.2.41 pnl()

```
void pnl (
            int reqId,
            double dailyPnL,
            double unrealizedPnL,
            double realizedPnL )
```

receives PnL updates in real time for the daily PnL and the total unrealized PnL for an account

Parameters

reqId	
dailyPnL	dailyPnL updates for the account in real time
unrealizedPnL	total unRealized PnL updates for the account in real time

See also

EClient::reqPnL

26.15.2.42 pnlSingle()

```
void pnlSingle (
            int reqId,
            int pos,
            double dailyPnL,
            double unrealizedPnL,
            double realizedPnL,
            double value )
```

receives real time updates for single position daily PnL values

Parameters

reqId	
pos	current size of the position
dailyPnL	dailyPnL for the position
unrealizedPnL	total unrealized PnL for the position (since inception) updating in real time
value	current market value of the position

See also

EClient::reqSinglePnL

26.15.2.43 position()

```
void position (
             string account,
              Contract contract,
             double pos,
             double avgCost )
```

provides the portfolio's open positions.

Parameters

account	the account holding the position.
contract	the position's **Contract**
pos	the number of positions held. avgCost the average cost of the position.

See also

positionEnd (p. **??**), **EClientSocket::reqPositions**

26.15.2.44 positionEnd()

```
void positionEnd ( )
```

Indicates all the positions have been transmitted.

See also

position (p. **??**), **EClient::reqPositions**

26.15.2.45 positionMulti()

```
void positionMulti (
             int requestId,
             string account,
             string modelCode,
              Contract contract,
             double pos,
             double avgCost )
```

provides the portfolio's open positions.

Parameters

requestId	the id of request
account	the account holding the position.
modelCode	the model code holding the position.
contract	the position's **Contract**
pos	the number of positions held.
avgCost	the average cost of the position.

See also

positionMultiEnd (p. **??**), **EClientSocket::reqPositionsMulti**

26.15.2.46 positionMultiEnd()

```
void positionMultiEnd (
            int requestId )
```

Indicates all the positions have been transmitted.

See also

positionMulti (p. **??**), **EClient::reqPositionsMulti**

26.15.2.47 realtimeBar()

```
void realtimeBar (
            int reqId,
            long time,
            double open,
            double high,
            double low,
            double close,
            long volume,
            double WAP,
            int count )
```

updates the real time 5 seconds bars

Parameters

reqId	the request's identifier
date	the bar's date and time (either as a yyyymmss hh:mm:ss formatted string or as system time according to the request)
open	the bar's open point
high	the bar's high point
low	the bar's low point
close	the bar's closing point
volume	the bar's traded volume (only returned for TRADES data)
WAP	the bar's Weighted Average Price rounded to minimum increment (only available for TRADES).
count	the number of trades during the bar's timespan (only available for TRADES).

See also

EClientSocket::reqRealTimeBars

26.15.2.48 receiveFA()

```
void receiveFA (
            int faDataType,
            string faXmlData )
```

receives the Financial Advisor's configuration available in the TWS

Parameters

faDataType	one of: 1. Groups: offer traders a way to create a group of accounts and apply a single allocation method to all accounts in the group. 2. Profiles: let you allocate shares on an account-by-account basis using a predefined calculation value. 3. Account Aliases: let you easily identify the accounts by meaningful names rather than account numbers.
faXmlData	the xml-formatted configuration

See also

EClientSocket::requestFA (p. **??**), **EClientSocket::replaceFA**

26.15.2.49 rerouteMktDataReq()

```
void rerouteMktDataReq (
            int reqId,
            int conId,
            string exchange )
```

- returns conId and exchange for CFD market data request re-route

Parameters

reqId	
conId	of the underlying instrument which has market data
exchange	of the underlying

26.15.2.50 rerouteMktDepthReq()

```
void rerouteMktDepthReq (
            int reqId,
            int conId,
            string exchange )
```

returns the conId and exchange for an underlying contract when a request is made for level 2 data for an instrument which does not have data in IB's database. For example stock CFDs and index CFDs.

Parameters

reqId	
conId	
exchange	

26.15.2.51 scannerData()

```
void scannerData (
            int reqId,
            int rank,
             ContractDetails contractDetails,
            string distance,
            string benchmark,
            string projection,
            string legsStr )
```

provides the data resulting from the market scanner request.

Parameters

reqid	the request's identifier.
rank	the ranking within the response of this bar.
contractDetails	the data's **ContractDetails**
distance	according to query.
benchmark	according to query.
projection	according to query.
legStr	describes the combo legs when the scanner is returning EFP

See also

scannerParameters (p. **??**), **scannerDataEnd** (p. **??**), **EClientSocket::reqScannerSubscription**

26.15.2.52 scannerDataEnd()

```
void scannerDataEnd (
            int reqId )
```

Indicates the scanner data reception has terminated.

Parameters

reqId	the request's identifier

See also

scannerParameters (p. **??**), **scannerData** (p. **??**), **EClientSocket::reqScannerSubscription**

26.15.2.53 scannerParameters()

```
void scannerParameters (
            string xml )
```

provides the xml-formatted parameters available from TWS market scanners (not all available in API).

Parameters

xml	the xml-formatted string with the available parameters.

See also

scannerData (p. **??**), **EClientSocket::reqScannerParameters**

26.15.2.54 securityDefinitionOptionParameter()

```
void securityDefinitionOptionParameter (
            int reqId,
            string exchange,
            int underlyingConId,
            string tradingClass,
            string multiplier,
            HashSet< string > expirations,
            HashSet< double > strikes )
```

returns the option chain for an underlying on an exchange specified in reqSecDefOptParams There will be multiple callbacks to securityDefinitionOptionParameter if multiple exchanges are specified in reqSecDefOptParams

Parameters

reqId	ID of the request initiating the callback
underlyingConId	The conID of the underlying security
tradingClass	the option trading class
multiplier	the option multiplier
expirations	a list of the expiries for the options of this underlying on this exchange
strikes	a list of the possible strikes for options of this underlying on this exchange

See also

EClient::reqSecDefOptParams

26.15.2.55 securityDefinitionOptionParameterEnd()

```
void securityDefinitionOptionParameterEnd (
            int reqId )
```

called when all callbacks to securityDefinitionOptionParameter are complete

Parameters

reqId	the ID used in the call to securityDefinitionOptionParameter

See also

securityDefinitionOptionParameter (p. **??**), **EClient::reqSecDefOptParams**

26.15.2.56 smartComponents()

```
void smartComponents (
            int reqId,
            Dictionary< int, KeyValuePair< string, char >> theMap )
```

bit number to exchange + exchange abbreviation dictionary

Parameters

reqId	
theMap	sa **EClient::reqSmartComponents**

26.15.2.57 softDollarTiers()

```
void softDollarTiers (
            int reqId,
             SoftDollarTier [] tiers )
```

called when receives Soft Dollar Tier configuration information

Parameters

reqId	The request ID used in the call to **EClient::reqSoftDollarTiers**
tiers	Stores a list of **SoftDollarTier** that contains all Soft Dollar Tiers information

See also

EClient::reqSoftDollarTiers

26.15.2.58 symbolSamples()

```
void symbolSamples (
            int reqId,
            ContractDescription [] contractDescriptions )
```

returns array of sample contract descriptions

Parameters

ContractDescription[ ]	

See also

EClient::reqMatchingSymbols

26.15.2.59 tickByTickAllLast()

```
void tickByTickAllLast (
            int reqId,
            int tickType,
            long time,
            double price,
            int size,
             TickAttrib attribs,
            string exchange,
            string specialConditions )
```

returns "Last" or "AllLast" tick-by-tick real-time tick

Parameters

reqId	- unique identifier of the request
tickType	- tick-by-tick real-time tick type: "Last" or "AllLast"
time	- tick-by-tick real-time tick timestamp
price	- tick-by-tick real-time tick last price

Parameters

size	- tick-by-tick real-time tick last size
attribs	- tick-by-tick real-time tick attribs (bit 0 - past limit, bit 1 - unreported)
exchange	- tick-by-tick real-time tick exchange
specialConditions	- tick-by-tick real-time tick special conditions

See also

EClient::reqTickByTickData

26.15.2.60 tickByTickBidAsk()

```
void tickByTickBidAsk (
            int reqId,
            long time,
            double bidPrice,
            double askPrice,
            int bidSize,
            int askSize,
             TickAttrib attribs )
```

returns "BidAsk" tick-by-tick real-time tick

Parameters

reqId	- unique identifier of the request
time	- tick-by-tick real-time tick timestamp
bidPrice	- tick-by-tick real-time tick bid price
askPrice	- tick-by-tick real-time tick ask price
bidSize	- tick-by-tick real-time tick bid size
askSize	- tick-by-tick real-time tick ask size
attribs	- tick-by-tick real-time tick attribs (bit 0 - bid past low, bit 1 - ask past high)

See also

EClient::reqTickByTickData

26.15.2.61 tickByTickMidPoint()

```
void tickByTickMidPoint (
            int reqId,
            long time,
            double midPoint )
```

returns "MidPoint" tick-by-tick real-time tick

Parameters

reqId	- unique identifier of the request
time	- tick-by-tick real-time tick timestamp
midPoint	- tick-by-tick real-time tick mid point

See also

EClient::reqTickByTickData

26.15.2.62 tickEFP()

```
void tickEFP (
            int tickerId,
            int tickType,
            double basisPoints,
            string formattedBasisPoints,
            double impliedFuture,
            int holdDays,
            string futureLastTradeDate,
            double dividendImpact,
            double dividendsToLastTradeDate )
```

Exchange for Physicals.

Parameters

tickerId	The request's identifier.
tickType	The type of tick being received.
basisPoints	Annualized basis points, which is representative of the financing rate that can be directly compared to broker rates.
formattedBasisPoints	Annualized basis points as a formatted string that depicts them in percentage form.
impliedFuture	The implied Futures price.
holdDays	The number of hold days until the lastTradeDate of the EFP.
futureLastTradeDate	The expiration date of the single stock future.
dividendImpact	The dividend impact upon the annualized basis points interest rate.
dividendsToLastTradeDate	The dividends expected until the expiration of the single stock future.

26.15.2.63 tickGeneric()

```
void tickGeneric (
            int tickerId,
            int field,
            double value )
```

Market data callback.

Parameters

tickerId	the request's unique identifier.
field	the type of tick being received.
value	

26.15.2.64 tickNews()

```
void tickNews (
            int tickerId,
            long timeStamp,
            string providerCode,
            string articleId,
            string headline,
            string extraData )
```

ticks with news headline

Parameters

tickerId	
timeStamp	
providerCode	
articleId	
headline	
extraData	

See also

EClient::reqMktData

26.15.2.65 tickOptionComputation()

```
void tickOptionComputation (
            int tickerId,
            int field,
            double impliedVolatility,
            double delta,
            double optPrice,
            double pvDividend,
            double gamma,
            double vega,
            double theta,
            double undPrice )
```

Receive's option specific market data. This method is called when the market in an option or its underlier moves. TWS's option model volatilities, prices, and deltas, along with the present value of dividends expected on that options underlier are received.

Parameters

tickerId	the request's unique identifier.
field	Specifies the type of option computation. Pass the field value into TickType.getField(int tickType) to retrieve the field description. For example, a field value of 13 will map to modelOptComp, etc. 10 = Bid 11 = Ask 12 = Last
impliedVolatility	the implied volatility calculated by the TWS option modeler, using the specified tick type value.
delta	the option delta value.
optPrice	the option price.
pwDividend	the present value of dividends expected on the option's underlying.
gamma	the option gamma value.
vega	the option vega value.
theta	the option theta value.
undPrice	the price of the underlying.

See also

TickType, **tickSize** (p. **??**), **tickPrice** (p. **??**), **tickEFP** (p. **??**), **tickGeneric** (p. **??**), **tickString** (p. **??**), **tickSnapshotEnd** (p. **??**), **marketDataType** (p. **??**), **EClientSocket::reqMktData**

26.15.2.66 tickPrice()

```
void tickPrice (
            int tickerId,
            int field,
            double price,
             TickAttrib attribs )
```

Market data tick price callback. Handles all price related ticks. Every tickPrice callback is followed by a tickSize. A tickPrice value of -1 or 0 followed by a tickSize of 0 indicates there is no data for this field currently available, whereas a tickPrice with a positive tickSize indicates an active quote of 0 (typically for a combo contract).

Parameters

tickerId	the request's unique identifier.
field	the type of the price being received (i.e. ask price).
price	the actual price.
attribs	an **TickAttrib** object that contains price attributes such as **TickAttrib::CanAutoExecute** (p. **??**), **TickAttrib::PastLimit** and **TickAttrib::PreOpen** (p. **??**).

See also

TickType, **tickSize** (p. **??**), **tickString** (p. **??**), **tickEFP** (p. **??**), **tickGeneric** (p. **??**), **tickOptionComputation** (p. **??**), **tickSnapshotEnd** (p. **??**), **marketDataType** (p. **??**), **EClientSocket::reqMktData**

26.15.2.67 tickReqParams()

```
void tickReqParams (
            int tickerId,
            double minTick,
            string bboExchange,
            int snapshotPermissions )
```

tick with BOO exchange and snapshot permissions

Parameters

tickerId	
minTick	
bboExchange	
snapshotPermissions	sa **EClient::reqMktData**

26.15.2.68 tickSize()

```
void tickSize (
            int tickerId,
            int field,
            int size )
```

Market data tick size callback. Handles all size-related ticks.

Parameters

tickerId	the request's unique identifier.
field	the type of size being received (i.e. bid size)
size	the actual size. US stocks have a multiplier of 100.

See also

TickType, **tickPrice** (p. **??**), **tickString** (p. **??**), **tickEFP** (p. **??**), **tickGeneric** (p. **??**), **tickOptionComputation** (p. **??**), **tickSnapshotEnd** (p. **??**), **marketDataType** (p. **??**), **EClientSocket::reqMktData**

26.15.2.69 tickString()

```
void tickString (
            int tickerId,
            int field,
            string value )
```

Market data callback. Every tickPrice is followed by a tickSize. There are also independent tickSize callbacks anytime the tickSize changes, and so there will be duplicate tickSize messages following a tickPrice.

Parameters

tickerId	the request's unique identifier.
field	the type of the tick being received

See also

TickType, **tickSize** (p. **??**), **tickPrice** (p. **??**), **tickEFP** (p. **??**), **tickGeneric** (p. **??**), **tickOptionComputation** (p. **??**), **tickSnapshotEnd** (p. **??**), **marketDataType** (p. **??**), **EClientSocket::reqMktData**

26.15.2.70 updateAccountTime()

```
void updateAccountTime (
            string timestamp )
```

Receives the last time on which the account was updated.

Parameters

timestamp	the last update system time.

See also

updatePortfolio (p. **??**), **accountDownloadEnd** (p. **??**), **updateAccountValue** (p. **??**), **EClientSocket::reqAccountUpdates**

26.15.2.71 updateAccountValue()

```
void updateAccountValue (
            string key,
            string value,
            string currency,
            string accountName )
```

Receives the subscribed account's information. Only one account can be subscribed at a time. After the initial callback to updateAccountValue, callbacks only occur for values which have changed. This occurs at the time of a position change, or every 3 minutes at most. This frequency cannot be adjusted.

Parameters

key	the value being updated. • AccountCode — The account ID number • AccountOrGroup — "All" to return account summary data for all accounts, or set to a specific Advisor Account Group name that has already been created in TWS Global Configuration • AccountReady — For internal use only • AccountType — Identifies the IB account structure • AccruedCash — Total accrued cash value of stock, commodities and securities • AccruedCash-C — Reflects the current's month accrued debit and credit interest to date, updated daily in commodity segment • AccruedCash-S — Reflects the current's month accrued debit and credit interest to date, updated daily in security segment • AccruedDividend — Total portfolio value of dividends accrued • AccruedDividend-C — Dividends accrued but not paid in commodity segment • AccruedDividend-S — Dividends accrued but not paid in security segment • AvailableFunds — This value tells what you have available for trading • AvailableFunds-C — Net Liquidation Value - Initial Margin • AvailableFunds-S — Equity with Loan Value - Initial Margin • Billable — Total portfolio value of treasury bills • Billable-C — Value of treasury bills in commodity segment • Billable-S — Value of treasury bills in security segment • BuyingPower — Cash Account: Minimum (Equity with Loan Value, Previous Day Equity with Loan Value)-Initial Margin, Standard Margin Account: Minimum (Equity with Loan Value, Previous Day Equity with Loan Value) - Initial Margin ∗4 • CashBalance — Cash recognized at the time of trade + futures PNL • CorporateBondValue — Value of non-Government bonds such as corporate bonds and municipal bonds • Currency — Open positions are grouped by currency • Cushion — Excess liquidity as a percentage of net liquidation value • DayTradesRemaining — Number of Open/Close trades one could do before Pattern Day Trading is detected • DayTradesRemainingT+1 — Number of Open/Close trades one could do tomorrow before Pattern Day Trading is detected • DayTradesRemainingT+2 — Number of Open/Close trades one could do two days from today before Pattern Day Trading is detected • DayTradesRemainingT+3 — Number of Open/Close trades one could do three days from today before Pattern Day Trading is detected • DayTradesRemainingT+4 — Number of Open/Close trades one could do four days from today before Pattern Day Trading is detected • EquityWithLoanValue — Forms the basis for determining whether a client has the necessary assets to either initiate or maintain security positions
	• EquityWithLoanValue-C — Cash account: Total cash value + commodities option value - futures maintenance margin requirement + minimum (0, futures PNL) Margin account: Total cash value + commodities option value - futures maintenance margin requirement

Parameters

value	up-to-date value
currency	the currency on which the value is expressed.
accountName	the account

See also

updatePortfolio (p. **??**), **updateAccountTime** (p. **??**), **accountDownloadEnd** (p. **??**), **EClientSocket::req↩AccountUpdates**

26.15.2.72 updateMktDepth()

```
void updateMktDepth (
            int tickerId,
            int position,
            int operation,
            int side,
            double price,
            int size )
```

Returns the order book.

Parameters

tickerId	the request's identifier
position	the order book's row being updated
operation	how to refresh the row: 0 = insert (insert this new order into the row identified by 'position')· 1 = update (update the existing order in the row identified by 'position')· 2 = delete (delete the existing order at the row identified by 'position').
side	0 for ask, 1 for bid
price	the order's price
size	the order's size

See also

updateMktDepthL2 (p. **??**), **EClientSocket::reqMarketDepth**

26.15.2.73 updateMktDepthL2()

```
void updateMktDepthL2 (
            int tickerId,
            int position,
            string marketMaker,
            int operation,
```

```
            int side,
            double price,
            int size )
```

Returns the order book.

Parameters

tickerId	the request's identifier
position	the order book's row being updated
marketMaker	the exchange holding the order
operation	how to refresh the row: 0 - insert (insert this new order into the row identified by 'position')· 1 - update (update the existing order in the row identified by 'position')· 2 - delete (delete the existing order at the row identified by 'position').
side	0 for ask, 1 for bid
price	the order's price
size	the order's size

See also

updateMktDepth (p. **??**), **EClientSocket::reqMarketDepth**

26.15.2.74 updateNewsBulletin()

```
void updateNewsBulletin (
            int msgId,
            int msgType,
            String message,
            String origExchange )
```

provides IB's bulletins

Parameters

msgId	the bulletin's identifier
msgType	one of: 1 - Regular news bulletin 2 - Exchange no longer available for trading 3 - Exchange is available for trading
message	the message
origExchange	the exchange where the message comes from.

26.15.2.75 updatePortfolio()

```
void updatePortfolio (
             Contract contract,
            double position,
```

```
double marketPrice,
double marketValue,
double averageCost,
double unrealizedPNL,
double realizedPNL,
string accountName )
```

Receives the subscribed account's portfolio. This function will receive only the portfolio of the subscribed account. If the portfolios of all managed accounts are needed, refer to EClientSocket::reqPosition After the initial callback to updatePortfolio, callbacks only occur for positions which have changed.

Parameters

contract	the **Contract** for which a position is held.
position	the number of positions held.
marketPrice	instrument's unitary price
marketValue	total market value of the instrument.

See also

updateAccountTime (p. **??**), **accountDownloadEnd** (p. **??**), **updateAccountValue** (p. **??**), **EClientSocket::reqAccountUpdates**

26.16 Execution Class Reference

Class describing an order's execution.

Public Member Functions

- **Execution** (int orderId, int clientId, String execId, String time, String acctNumber, String exchange, String side, double shares, double price, int permId, int liquidation, double cumQty, double avgPrice, String orderRef, String evRule, double evMultiplier, String modelCode, **Liquidity** lastLiquidity)
- override bool **Equals** (Object p_other)

Properties

- int **OrderId** `[get, set]`

  The API client's order Id. May not be unique to an account.
- int **ClientId** `[get, set]`

  The API client identifier which placed the order which originated this execution.
- string **ExecId** `[get, set]`

  The execution's unique identifier. Each partial fill has a separate ExecId.
- string **Time** `[get, set]`

  The execution's server time.
- string **AcctNumber** `[get, set]`

  The account to which the order was allocated.
- string **Exchange** `[get, set]`

  The exchange where the execution took place.
- string **Side** `[get, set]`

Specifies if the transaction was buy or sale BOT for bought, SLD for sold.

- double **Shares** `[get, set]`

  The number of shares filled.

- double **Price** `[get, set]`

  The order's execution price excluding commissions.

- int **PermId** `[get, set]`

  The TWS order identifier. The PermId can be 0 for trades originating outside IB.

- int **Liquidation** `[get, set]`

  Identifies whether an execution occurred because of an IB-initiated liquidation.

- double **CumQty** `[get, set]`

  Cumulative quantity. Used in regular trades, combo trades and legs of the combo.

- double **AvgPrice** `[get, set]`

  Average price. Used in regular trades, combo trades and legs of the combo. Does not include commissions.

- string **OrderRef** `[get, set]`

  The OrderRef is a user-customizable string that can be set from the API or TWS and will be associated with an order for its lifetime.

- string **EvRule** `[get, set]`

  The Economic Value Rule name and the respective optional argument. The two values should be separated by a colon. For example, aussieBond:YearsToExpiration=3. When the optional argument is not present, the first value will be followed by a colon.

- double **EvMultiplier** `[get, set]`

  Tells you approximately how much the market value of a contract would change if the price were to change by 1. It cannot be used to get market value by multiplying the price by the approximate multiplier.

- string **ModelCode** `[get, set]`

  model code

- **Liquidity LastLiquidity** `[get, set]`

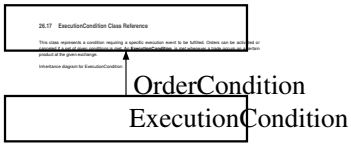
  The liquidity type of the execution. Requires TWS 968+ and API v973.05+. Python API specifically requires API v973.06+.

26.16.1 Detailed Description

Class describing an order's execution.

See also

ExecutionFilter (p. **??**), **CommissionReport**



Public Member Functions

- override string **ToString** ()
- override void **Deserialize** (IDecoder inStream)
- override void **Serialize** (BinaryWriter outStream)
- override bool **Equals** (object obj)
- override int **GetHashCode** ()

Protected Member Functions

- override bool **TryParse** (string cond)

Properties

- string **Exchange** `[get, set]`
  Exchange where the symbol needs to be traded.
- string **SecType** `[get, set]`
  Kind of instrument being monitored.
- string **Symbol** `[get, set]`
  Instrument's symbol.

26.17.1 Detailed Description

This class represents a condition requiring a specific execution event to be fulfilled. Orders can be activated or canceled if a set of given conditions is met. An **ExecutionCondition** is met whenever a trade occurs on a certain product at the given exchange.

26.18 ExecutionFilter Class Reference

when requesting executions, a filter can be specified to receive only a subset of them

Public Member Functions

- **ExecutionFilter** (int clientId, String acctCode, String time, String symbol, String secType, String exchange, String side)
- override bool **Equals** (Object other)

Properties

- int **ClientId** `[get, set]`
  The API client which placed the order.
- string **AcctCode** `[get, set]`
  The account to which the order was allocated to.
- string **Time** `[get, set]`
  Time from which the executions will be brough yyyymmdd hh:mm:ss Only those executions reported after the specified time will be returned.
- string **Symbol** `[get, set]`
  The instrument's symbol.
- string **SecType** `[get, set]`
  *The **Contract** (p. **??**)'s security's type (i.e. STK, OPT...)*
- string **Exchange** `[get, set]`
  The exchange at which the execution was produced.
- string **Side** `[get, set]`
  *The **Contract** (p. **??**)'s side (Put or Call).*

26.18.1 Detailed Description

when requesting executions, a filter can be specified to receive only a subset of them

See also

Contract (p. **??**), **Execution** (p. **??**), **CommissionReport**

26.19 HistoricalTick Class Reference

The historical tick's description. Used when requesting historical tick data with whatToShow = MIDPOINT.

Public Member Functions

- **HistoricalTick** (long time, double price, long size)

Properties

- long **Time** `[get]`
  The UNIX timestamp of the historical tick.
- double **Price** `[get]`
  The historical tick price.
- long **Size** `[get]`
  The historical tick size.

26.19.1 Detailed Description

The historical tick's description. Used when requesting historical tick data with whatToShow = MIDPOINT.

See also

EClient (p. **??**), **EWrapper**

26.20 HistoricalTickBidAsk Class Reference

The historical tick's description. Used when requesting historical tick data with whatToShow = BID_ASK.

Public Member Functions

- **HistoricalTickBidAsk** (long time, int mask, double priceBid, double priceAsk, long sizeBid, long sizeAsk)

Properties

- long **Time** [get]
  The UNIX timestamp of the historical tick.
- int **Mask** [get]
  Mask.
- double **PriceBid** [get]
  The bid price of the historical tick.
- double **PriceAsk** [get]
  The ask price of the historical tick.
- long **SizeBid** [get]
  The bid size of the historical tick.
- long **SizeAsk** [get]
  The ask size of the historical tick.

26.20.1 Detailed Description

The historical tick's description. Used when requesting historical tick data with whatToShow = BID_ASK.

See also

EClient (p. **??**), **EWrapper**

26.21 HistoricalTickLast Class Reference

The historical tick's description. Used when requesting historical tick data with whatToShow = TRADES.

Public Member Functions

- **HistoricalTickLast** (long time, int mask, double price, long size, string exchange, string specialConditions)

Properties

- long **Time** [get]
  The UNIX timestamp of the historical tick.
- int **Mask** [get]
  Mask.
- double **Price** [get]
  The last price of the historical tick.
- long **Size** [get]
  The last size of the historical tick.
- string **Exchange** [get]
  The source exchange of the historical tick.
- string **SpecialConditions** [get]
  The conditions of the historical tick. Refer to Trade Conditions page for more details: https://www.interactivebrokers.com/en/index.php?f=7235.

26.21.1 Detailed Description

The historical tick's description. Used when requesting historical tick data with whatToShow = TRADES.

See also

EClient (p. **??**), **EWrapper**

26.22 Liquidity Class Reference

Class describing the liquidity type of an execution.

Public Member Functions

- **Liquidity** (int p)
- override string **ToString** ()

Properties

- int **Value** `[get, set]`

  The value of the liquidity type.

26.22.1 Detailed Description

Class describing the liquidity type of an execution.

See also

Execution

26.23 Order Class Reference

The order's description.

Public Member Functions

- override bool **Equals** (Object p_other)

Static Public Attributes

- static int **CUSTOMER** = 0
- static int **FIRM** = 1
- static char **OPT_UNKNOWN** = '?'
- static char **OPT_BROKER_DEALER** = 'b'
- static char **OPT_CUSTOMER** = 'c'
- static char **OPT_FIRM** = 'f'
- static char **OPT_ISEMM** = 'm'
- static char **OPT_FARMM** = 'n'
- static char **OPT_SPECIALIST** = 'y'
- static int **AUCTION_MATCH** = 1
- static int **AUCTION_IMPROVEMENT** = 2
- static int **AUCTION_TRANSPARENT** = 3
- static string **EMPTY_STR** = ""

Properties

- int **OrderId** `[get, set]`

  The API client's order id.
- bool **Solicited** `[get, set]`
- int **ClientId** `[get, set]`

  The API client id which placed the order.
- int **PermId** `[get, set]`

  The Host order identifier.
- string **Action** `[get, set]`

  Identifies the side. Generally available values are BUY, SELL. Additionally, SSHORT, SLONG are available in some institutional-accounts only. For general account types, a SELL order will be able to enter a short position automatically if the order quantity is larger than your current long position. SSHORT is only supported for institutional account configured with Long/Short account segments or clearing with a separate account. SLONG is available in specially-configured institutional accounts to indicate that long position not yet delivered is being sold.

- double **TotalQuantity** `[get, set]`

  The number of positions being bought/sold.
- string **OrderType** `[get, set]`

  The order's type. Available Orders are at `https://www.interactivebrokers.com/en/software/api/apiguide/tabl`
  `_order_types.htm`.
- double **LmtPrice** `[get, set]`

  The LIMIT price. Used for limit, stop-limit and relative orders. In all other cases specify zero. For relative orders with no limit price, also specify zero.
- double **AuxPrice** `[get, set]`

  Generic field to contain the stop price for STP LMT orders, trailing amount, etc.
- string **Tif** `[get, set]`

  The time in force. Valid values are:
  DAY - Valid for the day only.
  GTC - Good until canceled. The order will continue to work within the system and in the marketplace until it executes or is canceled. GTC orders will be automatically be cancelled under the following conditions: If a corporate action on a security results in a stock split (forward or reverse), exchange for shares, or distribution of shares. If you do not log into your IB account for 90 days.
  At the end of the calendar quarter following the current quarter. For example, an order placed during the third quarter of 2011 will be canceled at the end of the first quarter of 2012. If the last day is a non-trading day, the cancellation will occur at the close of the final trading day of that quarter. For example, if the last day of the quarter is Sunday, the orders will be cancelled on the preceding Friday.
  Orders that are modified will be assigned a new "Auto Expire" date consistent with the end of the calendar quarter following the current quarter.

Orders submitted to IB that remain in force for more than one day will not be reduced for dividends. To allow adjustment to your order price on ex-dividend date, consider using a Good-Til-Date/Time (GTD) or Good-after-Time/Date (GAT) order type, or a combination of the two.
IOC - Immediate or Cancel. Any portion that is not filled as soon as it becomes available in the market is canceled.
GTD. - Good until Date. It will remain working within the system and in the marketplace until it executes or until the close of the market on the date specified
OPG - Use OPG to send a market-on-open (MOO) or limit-on-open (LOO) order.
FOK - If the entire Fill-or-Kill order does not execute as soon as it becomes available, the entire order is canceled.
DTC - Day until Canceled
.

- string **OcaGroup** `[get, set]`

  One-Cancels-All group identifier.

- int **OcaType** `[get, set]`

  Tells how to handle remaining orders in an OCA group when one order or part of an order executes. Valid values are:
  1 = Cancel all remaining orders with block.
  2 = Remaining orders are proportionately reduced in size with block.
  3 = Remaining orders are proportionately reduced in size with no block.
  If you use a value "with block" gives your order has overfill protection. This means that only one order in the group will be routed at a time to remove the possibility of an overfill.

- string **OrderRef** `[get, set]`

  The order reference. Intended for institutional customers only, although all customers may use it to identify the API client that sent the order when multiple API clients are running.

- bool **Transmit** `[get, set]`

  Specifies whether the order will be transmitted by TWS. If set to false, the order will be created at TWS but will not be sent.

- int **ParentId** `[get, set]`

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

- bool **BlockOrder** `[get, set]`

  If set to true, specifies that the order is an ISE Block order.

- bool **SweepToFill** `[get, set]`

  If set to true, specifies that the order is a Sweep-to-Fill order.

- int **DisplaySize** `[get, set]`

  The publicly disclosed order size, used when placing Iceberg orders.

- int **TriggerMethod** `[get, set]`

  Specifies how Simulated Stop, Stop-Limit and Trailing Stop orders are triggered. Valid values are:
  0 - The default value. The "double bid/ask" function will be used for orders for OTC stocks and US options. All other orders will used the "last" function.
  1 - use "double bid/ask" function, where stop orders are triggered based on two consecutive bid or ask prices.
  2 - "last" function, where stop orders are triggered based on the last price.
  3 double last function.
  4 bid/ask function.
  7 last or bid/ask function.
  8 mid-point function.
  .

- bool **OutsideRth** `[get, set]`

  If set to true, allows orders to also trigger or fill outside of regular trading hours.

- bool **Hidden** `[get, set]`

  If set to true, the order will not be visible when viewing the market depth. This option only applies to orders routed to the ISLAND exchange.

- string **GoodAfterTime** `[get, set]`

  Specifies the date and time after which the order will be active. Format: yyyymmdd hh:mm:ss {optional Timezone}.

- string **GoodTillDate** `[get, set]`

  The date and time until the order will be active. You must enter GTD as the time in force to use this string. The trade's "Good Till Date," format "YYYYMMDD hh:mm:ss (optional time zone)".

- bool **OverridePercentageConstraints** `[get, set]`

Overrides TWS constraints. Precautionary constraints are defined on the TWS Presets page, and help ensure tha tyour price and size order values are reasonable. Orders sent from the API are also validated against these safety constraints, and may be rejected if any constraint is violated. To override validation, set this parameter's value to True.

- string **Rule80A** [get, set]
  - *Individual = 'I'*
    Agency = 'A'
    AgentOtherMember = 'W'
    IndividualPTIA = 'J'
    AgencyPTIA = 'U'
    AgentOtherMemberPTIA = 'M'
    IndividualPT = 'K'
    AgencyPT = 'Y'
    AgentOtherMemberPT = 'N'
- bool **AllOrNone** [get, set]

  Indicates whether or not all the order has to be filled on a single execution.
- int **MinQty** [get, set]

  Identifies a minimum quantity order type.
- double **PercentOffset** [get, set]

  The percent offset amount for relative orders.
- double **TrailStopPrice** [get, set]

  Trail stop price for TRAILIMIT orders.
- double **TrailingPercent** [get, set]

  Specifies the trailing amount of a trailing stop order as a percentage. Observe the following guidelines when using the trailingPercent field:

  .
- string **FaGroup** [get, set]

  The Financial Advisor group the trade will be allocated to. Use an empty string if not applicable.
- string **FaProfile** [get, set]

  The Financial Advisor allocation profile the trade will be allocated to. Use an empty string if not applicable.
- string **FaMethod** [get, set]

  The Financial Advisor allocation method the trade will be allocated to. Use an empty string if not applicable.
- string **FaPercentage** [get, set]

  The Financial Advisor percentage concerning the trade's allocation. Use an empty string if not applicable.
- string **OpenClose** [get, set]

  For institutional customers only. Valid values are O (open), C (close). Available for institutional clients to determine if this order is to open or close a position. When Action = "BUY" and OpenClose = "O" this will open a new position. When Action = "BUY" and OpenClose = "C" this will close and existing short position.
- int **Origin** [get, set]

  The order's origin. Same as TWS "Origin" column. Identifies the type of customer from which the order originated. Valid values are 0 (customer), 1 (firm).
- int **ShortSaleSlot** [get, set]
  - *For institutions only. Valid values are: 1 (broker holds shares) or 2 (shares come from elsewhere).*
- string **DesignatedLocation** [get, set]

  Used only when shortSaleSlot is 2. For institutions only. Indicates the location where the shares to short come from. Used only when short sale slot is set to 2 (which means that the shares to short are held elsewhere and not with IB).
- int **ExemptCode** [get, set]

  Only available with IB Execution-Only accounts with applicable securities Mark order as exempt from short sale uptick rule.
- double **DiscretionaryAmt** [get, set]

  The amount off the limit price allowed for discretionary orders.
- bool **ETradeOnly** [get, set]

  Trade with electronic quotes.
- bool **FirmQuoteOnly** [get, set]

Trade with firm quotes.

- double **NbboPriceCap** [get, set]

  Maximum smart order distance from the NBBO.

- bool **OptOutSmartRouting** [get, set]

  Use to opt out of default SmartRouting for orders routed directly to ASX. This attribute defaults to false unless explicitly set to true. When set to false, orders routed directly to ASX will NOT use SmartRouting. When set to true, orders routed directly to ASX orders WILL use SmartRouting.

- int **AuctionStrategy** [get, set]

  – *For BOX orders only. Values include: 1 - match*
  2 - improvement
  3 - transparent

- double **StartingPrice** [get, set]

  The auction's starting price. For BOX orders only.

- double **StockRefPrice** [get, set]

  The stock's reference price. The reference price is used for VOL orders to compute the limit price sent to an exchange (whether or not Continuous Update is selected), and for price range monitoring.

- double **Delta** [get, set]

  The stock's Delta. For orders on BOX only.

- double **StockRangeLower** [get, set]

  The lower value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.

- double **StockRangeUpper** [get, set]

  The upper value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.

- double **Volatility** [get, set]

  The option price in volatility, as calculated by TWS' Option Analytics. This value is expressed as a percent and is used to calculate the limit price sent to the exchange.

- int **VolatilityType** [get, set]

  Values include:
  1 - Daily Volatility 2 - Annual Volatility.

- int **ContinuousUpdate** [get, set]

  Specifies whether TWS will automatically update the limit price of the order as the underlying price moves. VOL orders only.

- int **ReferencePriceType** [get, set]

  Specifies how you want TWS to calculate the limit price for options, and for stock range price monitoring. VOL orders only. Valid values include:
  1 - Average of NBBO
  2 - NBB or the NBO depending on the action and right.
  .

- string **DeltaNeutralOrderType** [get, set]

  Enter an order type to instruct TWS to submit a delta neutral trade on full or partial execution of the VOL order. VOL orders only. For no hedge delta order to be sent, specify NONE.

- double **DeltaNeutralAuxPrice** [get, set]

  Use this field to enter a value if the value in the deltaNeutralOrderType field is an order type that requires an Aux price, such as a REL order. VOL orders only.

- int **DeltaNeutralConId** [get, set]

  – *DOC_TODO*

- string **DeltaNeutralSettlingFirm** [get, set]

  – *DOC_TODO*

- string **DeltaNeutralClearingAccount** [get, set]

  – *DOC_TODO*

- string **DeltaNeutralClearingIntent** [get, set]

  – *DOC_TODO*

- string **DeltaNeutralOpenClose** [get, set]

  Specifies whether the order is an Open or a Close order and is used when the hedge involves a CFD and and the order is clearing away.
- bool **DeltaNeutralShortSale** [get, set]

  Used when the hedge involves a stock and indicates whether or not it is sold short.
- int **DeltaNeutralShortSaleSlot** [get, set]
  - *Has a value of 1 (the clearing broker holds shares) or 2 (delivered from a third party). If you use 2, then you must specify a deltaNeutralDesignatedLocation.*
- string **DeltaNeutralDesignatedLocation** [get, set]
  - *Used only when deltaNeutralShortSaleSlot = 2.*
- double **BasisPoints** [get, set]
  - *DOC_TODO For EFP orders only.*
- int **BasisPointsType** [get, set]
  - *DOC_TODO For EFP orders only.*
- int **ScaleInitLevelSize** [get, set]

  Defines the size of the first, or initial, order component. For Scale orders only.
- int **ScaleSubsLevelSize** [get, set]

  Defines the order size of the subsequent scale order components. For Scale orders only. Used in conjunction with scaleInitLevelSize().
- double **ScalePriceIncrement** [get, set]

  Defines the price increment between scale components. For Scale orders only. This value is compulsory.
- double **ScalePriceAdjustValue** [get, set]
  - *DOC_TODO For extended Scale orders.*
- int **ScalePriceAdjustInterval** [get, set]
  - *DOC_TODO For extended Scale orders.*
- double **ScaleProfitOffset** [get, set]
  - *DOC_TODO For extended scale orders.*
- bool **ScaleAutoReset** [get, set]
  - *DOC_TODO For extended scale orders.*
- int **ScaleInitPosition** [get, set]
  - *DOC_TODO For extended scale orders.*
- int **ScaleInitFillQty** [get, set]
  - *DOC_TODO For extended scale orders.*
- bool **ScaleRandomPercent** [get, set]
  - *DOC_TODO For extended scale orders.*
- string **HedgeType** [get, set]

  For hedge orders. Possible values include:
  D - delta
  B - beta
  F - FX
  P - Pair
  .
- string **HedgeParam** [get, set]
  - *DOC_TODO Beta = x for Beta hedge orders, ratio = y for Pair hedge order*
- string **Account** [get, set]

  The account the trade will be allocated to.
- string **SettlingFirm** [get, set]
  - *DOC_TODO Institutions only. Indicates the firm which will settle the trade.*
- string **ClearingAccount** [get, set]

  Specifies the true beneficiary of the order. For IBExecution customers. This value is required for FUT/FOP orders for reporting to the exchange.
- string **ClearingIntent** [get, set]

For exeuction-only clients to know where do they want their shares to be cleared at. Valid values are: IB, Away, and PTA (post trade allocation).

- string **AlgoStrategy** [get, set]

  The algorithm strategy. As of API verion 9.6, the following algorithms are supported:
  ArrivalPx - Arrival Price
  DarkIce - Dark Ice
  PctVol - Percentage of Volume
  Twap - TWAP (Time Weighted Average Price)
  Vwap - VWAP (Volume Weighted Average Price)
  For more information about IB's API algorithms, refer to https://www.interactivebrokers.↩com/en/software/api/apiguide/tables/ibalgo_parameters.htm.

- List< **TagValue** > **AlgoParams** [get, set]

  The list of parameters for the IB algorithm. For more information about IB's API algorithms, refer to https://www.interactivebrokers.com/en/software/api/apiguide/tables/ibalgo↩_parameters.htm.

- bool **WhatIf** [get, set]

  Allows to retrieve the commissions and margin information. When placing an order with this attribute set to true, the order will not be placed as such. Instead it will used to request the commissions and margin information that would result from this order.

- string **AlgoId** [get, set]
- bool **NotHeld** [get, set]

  Orders routed to IBDARK are tagged as "post only" and are held in IB's order book, where incoming SmartRouted orders from other IB customers are eligible to trade against them. For IBDARK orders only.

- List< **TagValue** > **SmartComboRoutingParams** [get, set]

  Advanced parameters for Smart combo routing.
  These features are for both guaranteed and nonguaranteed combination orders routed to Smart, and are available based on combo type and order type. SmartComboRoutingParams is similar to AlgoParams in that it makes use of tag/value pairs to add parameters to combo orders.
  Make sure that you fully understand how Advanced Combo Routing works in TWS itself first: https://www.↩interactivebrokers.com/en/software/tws/usersguidebook/specializedorderentry/advanced↩_combo_routing.htm
  The parameters cover the following capabilities:

- List< **OrderComboLeg** > **OrderComboLegs** [get, set]

  List of Per-leg price following the same sequence combo legs are added. The combo price must be left unspecified when using per-leg prices.

- List< **TagValue** > **OrderMiscOptions** [get, set]
  - *DOC_TODO*

- string **ActiveStartTime** [get, set]

  for GTC orders.

- string **ActiveStopTime** [get, set]

  for GTC orders.

- string **ScaleTable** [get, set]

  Used for scale orders.

- string **ModelCode** [get, set]

  model code

- string **ExtOperator** [get, set]

  This is a regulartory attribute that applies to all US Commodity (Futures) Exchanges, provided to allow client to comply with CFTC Tag 50 Rules.

- double **CashQty** [get, set]

  The native cash quantity.

- string **Mifid2DecisionMaker** [get, set]

  Identifies a person as the responsible party for investment decisions within the firm. Orders covered by MiFID 2 (Markets in Financial Instruments Directive 2) must include either Mifid2DecisionMaker or Mifid2DecisionAlgo field (but not both). Requires TWS 969+.

- string **Mifid2DecisionAlgo** [get, set]

Identifies the algorithm responsible for investment decisions within the firm. Orders covered under MiFID 2 must include either Mifid2DecisionMaker or Mifid2DecisionAlgo, but cannot have both. Requires TWS 969+.

- string **Mifid2ExecutionTrader** [get, set]

  For MiFID 2 reporting; identifies a person as the responsible party for the execution of a transaction within the firm. Requires TWS 969+.

- string **Mifid2ExecutionAlgo** [get, set]

  For MiFID 2 reporting; identifies the algorithm responsible for the execution of a transaction within the firm. Requires TWS 969+.

- bool **DontUseAutoPriceForHedge** [get, set]

  Don't use auto price for hedge.

- bool **RandomizeSize** [get, set]
  - *DOC_TODO*

- bool **RandomizePrice** [get, set]
  - *DOC_TODO*

- int **ReferenceContractId** [get, set]

  Pegged-to-benchmark orders: this attribute will contain the conId of the contract against which the order will be pegged.

- bool **IsPeggedChangeAmountDecrease** [get, set]

  Pegged-to-benchmark orders: indicates whether the order's pegged price should increase or decreases.

- double **PeggedChangeAmount** [get, set]

  Pegged-to-benchmark orders: amount by which the order's pegged price should move.

- double **ReferenceChangeAmount** [get, set]

  Pegged-to-benchmark orders: the amount the reference contract needs to move to adjust the pegged order.

- string **ReferenceExchange** [get, set]

  Pegged-to-benchmark orders: the exchange against which we want to observe the reference contract.

- string **AdjustedOrderType** [get, set]

  Adjusted Stop orders: the parent order will be adjusted to the given type when the adjusted trigger price is penetrated.

- double **TriggerPrice** [get, set]
  - *DOC_TODO*

- double **LmtPriceOffset** [get, set]
  - *DOC_TODO*

- double **AdjustedStopPrice** [get, set]

  Adjusted Stop orders: specifies the stop price of the adjusted (STP) parent.

- double **AdjustedStopLimitPrice** [get, set]

  Adjusted Stop orders: specifies the stop limit price of the adjusted (STPL LMT) parent.

- double **AdjustedTrailingAmount** [get, set]

  Adjusted Stop orders: specifies the trailing amount of the adjusted (TRAIL) parent.

- int **AdjustableTrailingUnit** [get, set]

  Adjusted Stop orders: specifies where the trailing unit is an amount (set to 0) or a percentage (set to 1)

- List< OrderCondition > **Conditions** [get, set]

  Conditions determining when the order will be activated or canceled.

- bool **ConditionsIgnoreRth** [get, set]

  Indicates whether or not conditions will also be valid outside Regular Trading Hours.

- bool **ConditionsCancelOrder** [get, set]

  Conditions can determine if an order should become active or canceled.

- **SoftDollarTier Tier** [get, set]

  Define the Soft Dollar Tier used for the order. Only provided for registered professional advisors and hedge and mutual funds.

26.23.1 Detailed Description

The order's description.

See also

Contract (p. **??**), **OrderComboLeg** (p. **??**), **OrderState**

26.23.2 Property Documentation

26.23.2.1 SmartComboRoutingParams

`List< TagValue> SmartComboRoutingParams [get], [set]`

Advanced parameters for Smart combo routing.
These features are for both guaranteed and nonguaranteed combination orders routed to Smart, and are available based on combo type and order type. SmartComboRoutingParams is similar to AlgoParams in that it makes use of tag/value pairs to add parameters to combo orders.
Make sure that you fully understand how Advanced Combo Routing works in TWS itself first: `https://www.interactivebrokers.com/en/software/tws/usersguidebook/specializedorderentry/advanced_combo_routing.htm`
The parameters cover the following capabilities:

- Non-Guaranteed - Determine if the combo order is Guaranteed or Non-Guaranteed.
  Tag = NonGuaranteed
  Value = 0: The order is guaranteed
  Value = 1: The order is non-guaranteed

- Select Leg to Fill First - User can specify which leg to be executed first.
  Tag = LeginPrio
  Value = -1: No priority is assigned to either combo leg
  Value = 0: Priority is assigned to the first leg being added to the comboLeg
  Value = 1: Priority is assigned to the second leg being added to the comboLeg
  Note: The LeginPrio parameter can only be applied to two-legged combo.

- Maximum Leg-In Combo Size - Specify the maximum allowed leg-in size per segment
  Tag = MaxSegSize
  Value = Unit of combo size

- Do Not Start Next Leg-In if Previous Leg-In Did Not Finish - Specify whether or not the system should attempt to fill the next segment before the current segment fills.
  Tag = DontLeginNext
  Value = 0: Start next leg-in even if previous leg-in did not finish
  Value = 1: Do not start next leg-in if previous leg-in did not finish

- Price Condition - Combo order will be rejected or cancelled if the leg market price is outside of the specified price range [CondPriceMin, CondPriceMax]
  Tag = PriceCondConid: The ContractID of the combo leg to specify price condition on
  Value = The ContractID
  Tag = CondPriceMin: The lower price range of the price condition
  Value = The lower price
  Tag = CondPriceMax: The upper price range of the price condition
  Value = The upper price

26.23.2.2 TrailingPercent

```
double TrailingPercent  [get], [set]
```

Specifies the trailing amount of a trailing stop order as a percentage. Observe the following guidelines when using the trailingPercent field:
.

- This field is mutually exclusive with the existing trailing amount. That is, the API client can send one or the other but not both.

- This field is read AFTER the stop price (barrier price) as follows: deltaNeutralAuxPrice stopPrice, trailing↩ Percent, scale order attributes

- The field will also be sent to the API in the openOrder message if the API client version is >= 56. It is sent after the stopPrice field as follows: stopPrice, trailingPct, basisPoint

26.24 OrderComboLeg Class Reference

Allows to specify a price on an order's leg.

Public Member Functions

- **OrderComboLeg** (double p_price)
- override bool **Equals** (Object other)

Properties

- double **Price** `[get, set]`

  The order's leg's price.

26.24.1 Detailed Description

Allows to specify a price on an order's leg.

See also

Order (p. **??**), **ComboLeg**

26.25 OrderState Class Reference

Provides an active order's current state.

Public Member Functions

- **OrderState** (string status, string initMarginBefore, string maintMarginBefore, string equityWithLoanBefore, string initMarginChange, string maintMarginChange, string equityWithLoanChange, string initMarginAfter, string maintMarginAfter, string equityWithLoanAfter, double commission, double minCommission, double maxCommission, string commissionCurrency, string warningText)
- override bool **Equals** (Object other)

Properties

- string **Status** `[get, set]`
  The order's current status.
- string **InitMarginBefore** `[get, set]`
  The account's current initial margin.
- string **MaintMarginBefore** `[get, set]`
  The account's current maintenance margin.
- string **EquityWithLoanBefore** `[get, set]`
  The account's current equity with loan.
- string **InitMarginChange** `[get, set]`
  The change of the account's initial margin.
- string **MaintMarginChange** `[get, set]`
  The change of the account's maintenance margin.
- string **EquityWithLoanChange** `[get, set]`
  The change of the account's equity with loan.
- string **InitMarginAfter** `[get, set]`
  The order's impact on the account's initial margin.
- string **MaintMarginAfter** `[get, set]`
  The order's impact on the account's maintenance margin.
- string **EquityWithLoanAfter** `[get, set]`
  Shows the impact the order would have on the account's equity with loan.
- double **Commission** `[get, set]`
  The order's generated commission.
- double **MinCommission** `[get, set]`
  The execution's minimum commission.
- double **MaxCommission** `[get, set]`
  The executions maximum commission.
- string **CommissionCurrency** `[get, set]`
  The generated commission currency.
- string **WarningText** `[get, set]`
  If the order is warranted, a descriptive message will be provided.

26.25.1 Detailed Description

Provides an active order's current state.

See also

Order

26.25.2 Property Documentation

26.25.2.1 CommissionCurrency

```
string CommissionCurrency  [get], [set]
```

The generated commission currency.

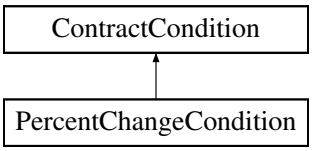
See also

CommissionReport

26.26 PercentChangeCondition Class Reference

Used with conditional orders to place or submit an order based on a percentage change of an instrument to the last close price.

Inheritance diagram for PercentChangeCondition:



Properties

- override string **Value** `[get, set]`
- double **ChangePercent** `[get, set]`

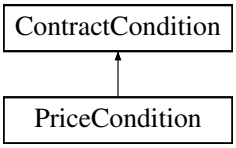
26.26.1 Detailed Description

Used with conditional orders to place or submit an order based on a percentage change of an instrument to the last close price.

26.27 PriceCondition Class Reference

Used with conditional orders to cancel or submit order based on price of an instrument.

Inheritance diagram for PriceCondition:



Public Member Functions

- override string **ToString** ()
- override bool **Equals** (object obj)
- override int **GetHashCode** ()
- override void **Deserialize** (IDecoder inStream)
- override void **Serialize** (BinaryWriter outStream)

Protected Member Functions

- override bool **TryParse** (string cond)

Properties

- override string **Value** `[get, set]`
- double **Price** `[get, set]`
- TriggerMethod **TriggerMethod** `[get, set]`

26.27.1 Detailed Description

Used with conditional orders to cancel or submit order based on price of an instrument.

26.28 ScannerSubscription Class Reference

Defines a market scanner request.

Properties

- int **NumberOfRows** [get, set]

  The number of rows to be returned for the query.
- string **Instrument** [get, set]

  The instrument's type for the scan. I.e. STK, FUT.HK, etc.
- string **LocationCode** [get, set]

  The request's location (STK.US, STK.US.MAJOR, etc).
- string **ScanCode** [get, set]

  Same as TWS Market Scanner's "parameters" field, for example: TOP_PERC_GAIN.
- double **AbovePrice** [get, set]

  Filters out Contracts which price is below this value.
- double **BelowPrice** [get, set]

  Filters out contracts which price is above this value.
- int **AboveVolume** [get, set]

  Filters out Contracts which volume is above this value.
- int **AverageOptionVolumeAbove** [get, set]

  Filters out Contracts which option volume is above this value.
- double **MarketCapAbove** [get, set]

  Filters out Contracts which market cap is above this value.
- double **MarketCapBelow** [get, set]

  Filters out Contracts which market cap is below this value.
- string **MoodyRatingAbove** [get, set]

  Filters out Contracts which Moody's rating is below this value.
- string **MoodyRatingBelow** [get, set]

  Filters out Contracts which Moody's rating is above this value.
- string **SpRatingAbove** [get, set]

  Filters out Contracts with a S&P rating below this value.
- string **SpRatingBelow** [get, set]

  Filters out Contracts with a S&P rating above this value.
- string **MaturityDateAbove** [get, set]

  Filter out Contracts with a maturity date earlier than this value.
- string **MaturityDateBelow** [get, set]

  Filter out Contracts with a maturity date older than this value.
- double **CouponRateAbove** [get, set]

  Filter out Contracts with a coupon rate lower than this value.
- double **CouponRateBelow** [get, set]

  Filter out Contracts with a coupon rate higher than this value.
- string **ExcludeConvertible** [get, set]

  Filters out Convertible bonds.
- string **ScannerSettingPairs** [get, set]

  For example, a pairing "Annual, true" used on the "top Option Implied Vol % Gainers" scan would return annualized volatilities.
- string **StockTypeFilter** [get, set]
  - *CORP = Corporation ADR = American Depositary Receipt ETF = Exchange Traded Fund REIT = Real Estate Investment Trust CEF = Closed End Fund*

26.28.1 Detailed Description

Defines a market scanner request.

26.29 SoftDollarTier Class Reference

A container for storing Soft Dollar Tier information.

Public Member Functions

- **SoftDollarTier** (string name, string value, string displayName)
- override bool **Equals** (object obj)
- override int **GetHashCode** ()
- override string **ToString** ()

Static Public Member Functions

- static bool **operator==** (**SoftDollarTier** left, **SoftDollarTier** right)
- static bool **operator!=** (**SoftDollarTier** left, **SoftDollarTier** right)

Properties

- string **Name** `[get]`

  The name of the Soft Dollar Tier.
- string **Value** `[get]`

  The value of the Soft Dollar Tier.
- string **DisplayName** `[get]`

  The display name of the Soft Dollar Tier.

26.29.1 Detailed Description

A container for storing Soft Dollar Tier information.

26.30 TagValue Class Reference

Convenience class to define key-value pairs.

Public Member Functions

- **TagValue** (string p_tag, string p_value)
- override bool **Equals** (Object other)

Properties

- string **Tag** `[get, set]`
- string **Value** `[get, set]`

26.30.1 Detailed Description

Convenience class to define key-value pairs.

26.31 TickAttrib Class Reference

Tick attributes that describes additional information for price ticks.

Properties

- bool **CanAutoExecute** `[get, set]`

  Specifies whether the price tick is available for automatic execution (1) or not (0)
- bool **PastLimit** `[get, set]`

  Indicates whether the bid price is lower than the day's lowest value or the ask price is higher than the highest ask.
- bool **PreOpen** `[get, set]`

  Indicates whether the bid/ask price tick is from pre-open session.
- bool **Unreported** `[get, set]`
- bool **BidPastLow** `[get, set]`
- bool **AskPastHigh** `[get, set]`

26.31.1 Detailed Description

Tick attributes that describes additional information for price ticks.

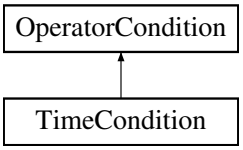
See also

EWrapper::tickPrice

26.32 TimeCondition Class Reference

Time condition used in conditional orders to submit or cancel orders at specified time.

Inheritance diagram for TimeCondition:



Public Member Functions

- override string **ToString** ()

Protected Member Functions

- override bool **TryParse** (string cond)

Properties

- override string **Value** `[get, set]`
- string **Time** `[get, set]`

  Time field used in conditional order logic. Valid format: YYYYMMDD HH:MM:SS.

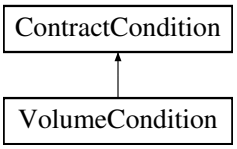
26.32.1 Detailed Description

Time condition used in conditional orders to submit or cancel orders at specified time.

26.33 VolumeCondition Class Reference

Used with conditional orders to submit or cancel an order based on a specified volume change in a security.

Inheritance diagram for VolumeCondition:



Properties

- override string **Value** `[get, set]`
- int **Volume** `[get, set]`

26.33.1 Detailed Description

Used with conditional orders to submit or cancel an order based on a specified volume change in a security.