TWS API Documentation

Introduction

The TWS API is a TCP Socket Protocol API based on connectivity to the Trader Workstation or IB Gateway. The API acts as an interface to retrieve and send data autonomously to Interactive Brokers. Interactive Brokers provides code systems in Python, Java, C++, C#, and VisualBasic.

The TWS API is a message protocol as its core, and any library that implements the TWS API, whether created by IB or someone else, is a tool to send and receive these messages over a TCP socket connection with the IB host platform (TWS or IB Gateway). As such the system can be tweaked and modified into any language of interest given the intention to translate the underlying decoder.

In short, a library written in any other languages must be sending and receiving the same data in the same format as any other conformant TWS API library, so users can look at the documentation for our libraries to see what a given request or response consists of (what it must include, in what form, etc.) and implement them in their own structure.
Our TWS API components are aimed at experienced professional developers willing to enhance the current TWS functionality. Before you use TWS API, please make sure you fully understand the concepts of OOP (https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/) and other Computer Science Concepts. 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

This guide references the Java, VB, C#, C++ and Python Testbed sample projects to demonstrate the TWS API functionality. 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.

Notes & Limitations

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!

Requests

The TWS/ IB Gateway is designed to **accept up to 50 messages per second coming from the client side**. Anything coming from the client application to the TWS/ IB Gateway 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 where all connected client applications to the same instance of TWS/ IB Gateway 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**.

In some use cases, if you plan to send more than 50 requests per second, some orders may be queued and delayed. For this scenario, please consider switching to FIX API.

For FIX API users in IB Gateway, the limitation is 250 messages per second.

For FIX API users without using IB Gateway or TWS, there is no limitation on messages per second, but less is better.

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.

Requirements

- A funded and opened IBKR Pro account
- The current Stable or Latest release of the TWS or IB Gateway
- The current Stable or Latest release of the TWS API
- A working knowledge of the programming language our **Testbed** sample projects are developed in.

Unset Values

Developers may often find a super-massive value returned from requests like market data, P&L information, and elsewhere. These are known as Unset values. Unset values are used throughout programming systems to indicate that a value is not available. Unset values are used in place of NULL characters to prevent any unexpected error be thrown in your code. Unset values are also used in place of values like 0 to avoid confusing viewers to believe they have an account balance of 0, or that an equity is worth $0.

An unset value is the maximum value of a given data type. So the Unset Double value will appear like 1.79769313486231570E308, which contains approximately 308 digits to intentionally appear extraneous.

Download TWS or IB Gateway

In order to use the TWS API, all customers must install either Trader Workstation or IB Gateway to connect the API to. Both downloads maintain the same level of usage and support; however, they both have equal benefits. For example, IB Gateway will be less resource intensive as there is no UI; however, the Trader Workstation has access all of the same information as the API, if users would like an interface to confirm data.

Download Trader Workstation

Download IB Gateway

It is recommended for API users to use offline TWS because TWS online version has automatic update. Please use same TWS version to make sure the TWS version and TWS API version are synced. These will help preventing version conflict issue.

Note:

1. For IBHK API users, it is commended to use IB Gateway instead of TWS. It is because all IBHK users cannot choose "Never Lock Trader Workstation" in TWS – Global Configuration – Lock and Exit. If there is inactivity, TWS will be locked and there will be API disconnection.

TWS Settings

Some TWS Settings affect API.

TWS Configuration For API Use

In TWS Global Configuration – API – Settings, there are many API settings. Please enable/disable some API settings based on your use case.

In this section, only the most important API settings for API connection and incident troubleshooting are covered.

Please:

- Enable "ActiveX and Socket Clients"
- Disable "Read-Only API"
- Enable "Create API message log file"
- Enable "Include market data in API log file"
- Change "Logging Level" to "Detail"

If you want to establish localhost API connection, please enable "Allow connection from localhost only".

If you want to connect TWS/ IB Gateway from a remote server, please disable "Allow connection from localhost only" and put the IP Address detected in "Accept incoming connection attempt from <IP Address>" into "Trusted IPs".

The IP Address is usually Private IPv4 Address.

"Trusted IPs" does not accept subnet (e.g. /27, /28). It only accepts single IP Addresses. In the following example, there is a remote computing cluster /27 which has 32 IP Addresses and the remote computing cluster will randomly assign one of the computing nodes to connect to TWS in every connection. To make this happen, every Private IPv4 Address of the subnet are put into the "Trusted IPs" (You can also exclude the first IP Network Address and the last IP Broadcast Address of the subnet).

For more details, please visit: https://www.interactivebrokers.com/campus/ibkr-api-page/twsapi-doc/#incoming-api-connections

Note:

1. In TWS, if you don' t put the remote IP Address into the "Trusted IPs" and you try to establish remote API connection, there will be a message "Accept incoming connection attempt from <IP Address>" which requires you to accept the connection or reject the connection. In IB Gateway, it is compulsory for IB Gateway users to put the remote IP Address into the "Trusted IPs". Otherwise, there will be no connection attempt detected.

Best Practice: Configure TWS / IB Gateway

The information listed below are not required or necessary in order to operate the TWS API. However, these steps include many references which can help improve the day to day usage of the TWS API that is not explicitly offered as a callable method within the API itself.

Memory Allocation

In TWS/ IB Gateway – "Global Configuration" – "General", you can adjust the **Memory Allocation (in MB)**".

This feature is to control how much memory your computer can assign to the TWS/ IB Gateway application. Usually, higher value allows users to have faster data returning speed.

Normally, it is recommended for API users to set 4000. However, it depends on your computer memory size because setting too high may cause High Memory Usage and application not responding.

For details, please visit: https://www.ibkrguides.com/traderworkstation/increase-tws-memory-size.htm

Note:

1. In IB Gateway Global Configuration – API – settings, there is no "Compatibility Mode: Send ISLAND for US stocks trading on NASDAQ". Specifying NASDAQ exchange in contract details may cause error if connecting to IB Gateway. For this error, please specify ISLAND exchange.

Daily & Weekly Reauthentication

It is compulsory for TWS/ IB Gateway users to **auto logoff/auto restart TWS/ IB Gateway daily** and **manually login TWS/ IB Gateway weekly** (Organizational client with 1M+USD in equity or large organizational client can request an exception from IB Gateway weekly reauthentication).

Daily Reauthentication

In TWS/ IB Gateway – "Global Configuration" – "Lock and Exit", you can choose the time that your TWS will be shut down.

For API users, it is recommended to choose "Never lock Trader Workstation" and "Auto restart".

Note:

1. IBHK users do not have "**Never lock Trader Workstation**" and "**Auto restart**" in TWS. It is suggested IBHK users to use IB Gateway for API connection because IB Gateway won't be locked due to activity and IBHK users can choose "**Auto restart**" in IB Gateway.

Weekly Reauthentication

The weekly authentication cycle starts on every Monday. If you receive `Login failed = Soft token=0 received instead of expected permanent for zdc1.ibllc.com:4001 (SSL)`, this means you need to manually login again to complete the weekly reauthentication task.

Request an exception from IB Gateway weekly reauthentication

To request an exception from IB Gateway weekly reauthentication, please submit a web ticket to "Technical Assistance" – "Connectivity". However, it is not guaranteed to accept the IB Gateway weekly reauthentication request.

In the web ticket, you need to provide:

1. IB Account Number
2. Username(s) of the IB Account number that need to be excluded from IB Gateway weekly reauthentication.
3. Is the IB Gateway on a Virtual Machine?
   - If yes, please provide which data center.
4. What kind of access is available for the machine (Only Remote Terminal or Physical Access to Console)?
   - If only physical access is available, which is the access control method?
5. Who is administering the machine?

Operating System Sleeping

OS Sleeping Mode also causes TWS/IB Gateway and API disconnection. It is strongly suggested to choose "**Never Sleep**".

For how to turn off Sleep Mode in Windows, please visit: https://support.microsoft.com/en-us/windows/how-to-adjust-power-and-sleep-settings-in-windows-26f623b5-4fcc-4194-863d-b824e5ea7679

For how to turn off Sleep Mode in MacOS, please visit: https://support.apple.com/en-hk/guide/mac-help/mchle41a6ccd/mac

Order Precautions

In TWS – "Global Configuration" – "API" – "Precautions", you can enable the following items to stop receiving the order submission messages.

- Enable "Bypass Order Precautions for API orders".
- Enable "Bypass Bond warning for API orders".
- Enable "Bypass negative yield to worst confirmation for API orders".
- Enable "Bypass Called Bond warning for API orders".
- Enable "Bypass "same action pair trade" warning for API orders".

- Enable "Bypass price-based volatility risk warning for API orders".
- Enable "Bypass US Stocks market data in shares warning for API orders".
- Enable "Bypass Redirect Order warning for Stock API orders".
- Enable "Bypass No Overfill Protection precaution for destinations where implied natively".

Connected IB Server Location in TWS

Each IB account has a pre-decided IB server. You can visit this link to know our IB servers' locations: https://www.interactivebrokers.com/download/IB-Host-and-Ports.pdf

Yet, all IB paper accounts are connected to US server by default and its location cannot be changed.

As IB servers in different regions have different scheduled server maintenance time (https://www.interactivebrokers.com/en/software/systemStatus.php), you may need to change the IB server location in order to avoid service downtime.

For checking your connected IB server location, you can go to TWS and click "Data" to see your Primary server. In the below image, the pre-decided IB server location is: cdc1.ibllc.com

If you want to change your live IB account server location in TWS, please submit a web ticket to "Technical Assistance" – "Connectivity" in order to request changing the IB server location.

In the web ticket, you need to provide:

1. Which account do you want to have IB server location change?
2. Which IB server location would you like to connect to?
   - TWS AMERICA – EAST (New York)
   - TWS AMERICA – CENTRAL (Chicago)
   - TWS Europe (Zurich)
   - TWS Asia (Hong Kong)
   - TWS Asia – CHINA (For mainland China users, if the account server is hosted in Hong Kong, they will automatically connect with the Shenzhen Gateway mcgw1.ibllc.com.cn)
3. Which IB scheduled maintenance time do you choose? (Recommended to choose the default schedule maintenance time of its own IB server location)
   - North America
   - Europe
   - Asia

After you submit the ticket, you will receive a web ticket reply which **require you to confirm and understand the migration request**.

Note:

1. For Internet users, as the connection between IB server and Exchange goes through a dedicated line, it is commonly recommended to choose a IB server location which is closer to your TWS location. For IB connection types, please visit: https://www.interactivebrokers.co.uk/en/software/connectionInterface.php
2. The pre-decided IB server location connected from TWS is different from the IB Server location connected from IB Client Portal and IBKR Mobile.
   - IB server location connected from TWS is pre-decided. You can submit a web ticket to request the IB server relocation for the TWS connection.
   - IB server location connected from Client Portal, IBKR Mobile is based on your nearest IB server location. You cannot request the IB server relocation for Client Portal and IBKR Mobile connections. However, OAuth CP API users can specify which server they want to connect to by themselves. For details, please visit: https://www.interactivebrokers.com/campus/ibkr-api-page/cpapi-v1/#oauth-base-url

SMART Algorithm

In TWS Global Configuration – Orders – Smart Routing, you can set your SMART order routing algorithm. For available SMART Routing via TWS API, please visit: https://www.interactivebrokers.com/campus/ibkr-api-page/contracts/#smart-routing

Allocation Setup (For Financial Advisors)

In TWS Global Configuration – Advisor Setup – Presets, you can need to choose Allocation Preference in order to avoid wrong allocation result.

Intelligent Order Resubmission

The TWS Setting listed in the Global Configuration under API -> Setting for **Maintain and resubmit orders when connection is restored**, is enabled by default in TWS 10.28 and above. When this setting is checked, all orders received while connectivity is lost will be saved and automatically resubmitted when connectivity is restored. Please note, if the Trader Workstation is closed during this time, the orders are deleted regardless of the setting.

Disconnect on Invalid Format

The TWS Setting listed in the Global Configuration under API -> Setting for **Maintain connection upon receiving incorrectly formatted fields**, is enabled by default in TWS 10.28 and above. For clients operating on Client Version 100 and above, users will not disconnect from fields with invalid value submissions when the setting is enabled.

Download the TWS API

Current LTS API version is 10.16 or above. It is STRONGLY RECOMMENDED for 10.16 version below API users to upgrade API.

TWS API Download Page

It is recommended for API users to use same TWS API version to make sure the TWS version and TWS API version are synced in order to prevent version conflict issue.

Running the Windows version of the API installer creates a directory "C:\\TWS API\" for the API source code in addition to automatically copying two files into the Windows directory for the 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 ActiveX control TWSLib.dll, C# API CSharpAPI.dll, and C++ API TwsSocketClient.dll. Starting in API version **973.07**, running the API installer is designed to install an ActiveX control TWSLib.dll, and TwsRtdServer control TwsRTDServer.dll which are compatible with both 32 and 64 bit applications.

It is important to know that the TWS API is **only** available through the interactivebrokers.github.io MSI or ZIP file. Any other resource, including pip, NuGet, or any other online repository is not hosted, endorsed, supported, or connected to Interactive Brokers. As such, updates to the installation should always be downloaded from the github directly.

How to install the TWS API Components on Mac / Unix

1. Download the IB API Stable for Mac/Unix zip file to your local machine
2. This will direct you to Interactive Brokers **API License Agreement**, please review it
3. Once you have clicked "***I Agree***", refer to the *Mac / Unix* section to download the API Software version of

your preference

1. This will download **twsapi_macunix.<Major Version>.<Minor Version>.zip** to your computer
   (where <Major Version> and <Minor Version> are the major and minor version numbers respectively)

2. Open Terminal (**Ctrl+Alt+T** on most distributions)
   *(On Mac press **Command+Space** to launch Spotlight, then type **terminal** and press **Return**)*

3. Navigate to the directory where the installer has been downloaded (normally it should be the Download folder within your home folder) and confirm the file is present
   $ cd ~/Downloads
   $ ls

1. Unzip the contents the installer into your home folder with the following command *(if prompted, enter your password)*:
   NOTE: replace the values 'n.m' with the name of your installed file.
   $ sudo unzip twsapi_macunix.n.m.zip -d $HOME/

2. To access the sample and source files, navigate to the **IBJts** directory and confirm the subfolders samples and source are present
   $ cd ~/IBJts
   $ ls

Note:

- When running "***python3 setup.py install***", you may get "***ModuleNotFoundError: No Module named 'setuptools'***". As "**setuptools**" is deprecated, please grant the write permission on the target folder (e.g. **source/pythonclient)** using "***sudo chmod -R 777***" in order to avoid "***error: could not create 'ibapi.egg-info': Permission denied***". After that, run "***python3 -m pip install .***"

TWS API Folder Files & Tools

TWS API Folder Files Explanation:

- **"API_VersionNum.txt"**

File Path: C:\TWS API\API_VersionNum.txt

You can check your API version in this file.

- **"IBSampleApp.exe"**

File Path: C:\TWS API\samples\CSharp\IBSampleApp\bin\Release\IBSampleApp.exe

You can manually use the IBSampleApp to test the API functions.

Unique Configurations

While all of the available Trader Workstation API default samples provide equivalent functionality, some languages have unique configurations that must be implemented in order to use our samples or program code with the underlying API.

Implementing the Intel Decimal Library for MacOS and Linux

Due to the malleability of the many Linux distributions including MacOS, Interactive Brokers is unable to provide a pre-built binary for the library. As such, users programming in C++ on a Linux machine must manually build the Intel® Decimal Floating-Point Math Library manually.

As described in the README file from the linked page, you can find the library's build steps within the ~/IntelRDFPMathLib20U2/LIBRARY/README file.

Download the Intel® Decimal Floating-Point Math Library

Updating The Python Interpreter

Python has a unique system for importing libraries into it's IDEs. This extends even further when it comes to virtual environments. In order to utilize Python code with the TWS API, you must run our setup file in order to import the code.

1. Open Command Prompt or Terminal

In order to update the Python IDE, these steps MUST be performed through Command Prompt or Terminal. This can not be done through an explorer interface.

As such, users should begin by launching their respective command line interface.

These samples will display Windows commands, though the procedure is identical on Windows, MacOS, and Linux.

2. Navigate to Python Source

Customers should then change their directory to `{TWS API}\source\pythonclient` .

It is then recommend to display the contents of the directory with "ls" for Unix, or "dir" for Windows users.

3. Run The setup.py File

Customers will now need to run the setup.py steps with the installation parameter. This can be done with the command: `python setup.py install`

4. Confirm Updates

After running the prior command, users should see a large block of text describing various values being updated and added to their system. It is important to confirm that the version installed on your system mirrors the build version displayed. This example represents 10.25; however, you may have a different version.

5. Confirm your installation

Finally, users should look to confirm their installation. The simplest way to do this is to confirm their version with pip. Typing this command should show the latest installed version on your system: `python -m pip show ibapi`

Implementing Visual Basic .NET

Our VB.NET code is provided for demonstration purposes only; there is no pure, standalone VB.NET-based TWS API library. Both our "VB_API_Sample" and the VB.NET "Testbed" projects included with our TWS API releases call the C# TWS API source. The provided VB.NET code only interfaces with the C# source. Please keep in mind that these samples are in VB.NET, not Visual Basic for Applications.

Third Party API Platforms

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 IBKR account structures. Always check first with the software vendor before opening a specific account type or converting an IBKR account type. For instance, many third party API platforms such as NinjaTrader and TradeNavigator are **not** compatible with IBKR linked account structures, so it is highly recommended to first check with the third party vendor before linking your IBKR accounts.
An ongoing list of common Third Party Connections are available within our documentation. This resource will also link out to connection guides detailing how a user can connect with a given platform.

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.

Troubleshooting & Support

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 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.

Understanding Order Precautions

By default, the Trader Workstation implements several precautionary settings that will notify customers of potential order risks to make sure users are well informed before transmitting orders. As a result, customers will typically need to acknowledge a precautionary message and manually transmit the orders through the Trader Workstation. These precautionary messages may be disabled if the user is comfortable and aware of the behavior they are disabling.

Disabling Warning Messages

1. Log in to the Trader Workstation
2. Open the Global Configuration by selecting the Cog Wheel icon in the top right corner
3. Navigate to the "Messages" section on the left.
4. **Carefully read each message before disabling it**. You can then disable the warning by unchecking the box on the right of the message description.

Modifying Precautionary Settings

1. Log in to the Trader Workstation
2. Open the Global Configuration by selecting the Cog Wheel icon in the top right corner
3. Navigate to the "Presets" section on the left
4. Select the instrument(s) you are trading
5. **Carefully read each setting before making changes to it**. You may modify the values inside the "Precautionary Settings" settings to be more or less restrictive. You may also set the value to '0' to disable the precaution entirely.

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.

API Logs

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. 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. Historical candlestick data is always recorded in the API log.

Note: Both the API and TWS logs are encrypted locally. The API logs can be decrypted for review from the associated TWS or IB Gateway session, just like the TWS logs, as shown in the section describing the Local location of logs.
Enabling creation of API logs

TWS:

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

IB Gateway:

1. Navigate to Configure → Settings → API → Settings
2. Check the box *Create API message log file*
3. Click Apply and Ok

TWS Log Files

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.

TWS:

1. Navigate to File/Edit → Global Configuration → API → Settings
2. Set *Logging Level* to *Detail*

IB Gateway:

1. Navigate to Configure → Settings → API → Settings
2. Set *Logging Level* to *Detail*

Local location of logs

Logs are stored in 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).

Due to privacy regulations, logs are **encrypted** before they are saved to disk. They can be decrypted from the associated TWS or IB Gateway session.

- In TWS: Classic TWS Help -> Troubleshooting -> Diagnostics -> TWS Logs.
- In IB Gateway, File -> Gateway Logs.

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.

To upload logs as a Windows user:

1. In TWS or IB Gateway, press CTRL+ALT+H to bring up the Upload Diagnostics window.
2. In the "reason" text field, please type the reason for your upload.
   - Alternatively, type "ATTENTION: " and then the ticket number you are working with, or the name of your customer service representative.
3. Find the small arrow in the upper right corner, click it and select "Advanced View"
4. Make sure "Full internal state of the application" is checked
5. Make sure "Include previous days logs and settings" is unchecked, unless the error happened on a prior day.
6. Click Submit

To upload logs as a Mac and Linux user:

1. In TWS or IB Gateway, press CMD+OPT+H to bring up the Upload Diagnostics window.
2. In the "reason" text field, please type the reason for your upload.
   - Alternatively, type "ATTENTION: " and then the ticket number you are working with, or the name of your customer service representative.
3. Find the small arrow in the upper right corner, click it and select "Advanced View"
4. Make sure "Full internal state of the application" is checked
5. Make sure "Include previous days logs and settings" is unchecked, unless the error happened on a prior day.
6. Click 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.

Exporting Logs

1. In TWS, navigate to Help menu >> Troubleshooting >> Diagnostics >> "API Logs" or "TWS Logs".
2. In IBG, both "API Logs" and "Gateway Logs" are accessible directly from the File menu.
3. Click "Export Today Logs..." to decrypt the logs and save them in plaintext (logs are stored encrypted on your local machine)

Reading Exported Logs

Each supported API language of the API contains a message file that translates a given number identifier into their corresponding request. The message identifier numbers used in the underlying wire protocol is the core of the TWS API.

The information on the right documents where each message reader file is located. The {TWS API} listed is the path to the primary TWS API or JTS folder created from the API installation.

Python | Java | C++ | C# | VB.NET

Both the Incoming and Outgoing message IDs are listed in one file.

{TWS API}\source\pythonclient\ibapi\messages.py

By default, this will be saved directly on the *C:* drive.
In our API logs, the direction of the message is indicated by the arrow at the beginning:

-> for incoming messages (TWS to client)

<- for outgoing messages (client to TWS)

Thus <- **3** (outgoing request of type 3) is a placeOrder request, and the subsequent incoming requests are:

-> **5** = openOrder response

-> **11** = executionData response

-> **59** = commissionReport response

Also note that the first openOrder response carries with it an orderStatus response in the same message. If that status were to change later, it would be delivered as a standalone message:

-> **3** = orderStatus response

How To Enable Debug Logging

Enabling DEBUG-level logging for the host platform (TWS or IBG, this does not affect API logs):

1. Navigate to the root TWS/IBG installation directory
2. Find jts.ini and open in text editor
3. Put debug=1 under the [Communication] section
4. Reboot TWS/IBG

Setting debug=1 has added benefits in TWS.

1. Debug=1 also allows you to enter conIds into a watchlist to resolve them into symbols. Type/paste the conId in an empty watchlist row, add IC (vertical bar, capital C) at the end, and press Enter. Example: 265598IC will resolve immediately to AAPL (exchange will be SMART where available, primary otherwise).
   - If the instrument is already present in the watchlist, nothing will happen.
2. Additional detail in the "Description" window for an instrument, normally available by right-clicking on an instrument in a watchlist and selecting Financial Instrument Info >> Description from the context menu. Debug=1 will add the conId, min order sizes, market rules (i.e., min price increments and thresholds), all available order types, and all available exchanges to this interface. Changing the behavior of TWS to bring up that Description window on double-click can make it easier to find.
   1. In TWS, go to Global Configuration >> Display >> Ticker Row
   2. Change "Double-click on Financial Instrument will" dropdown menu to "Open Contract Details"

Architecture

The IBApi.EWrapper interface is the mechanism through which the TWS delivers information to the API client application. By implementing this interface the client application will be able to receive and handle the

Python | Java | C++ | C# | VB.NET

information coming from the TWS. For further information on how to implement interfaces, refer to your programming language's documentation.

The class used to send messages to TWS is IBApi.EClientSocket. 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 an EWrapper implementation so they can be 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.

In conclusion:

- **EClient functions are for requesting data to IB server.**
- **EWrapper functions are for receiving data from IB server.**
- **Missing anyone of them cannot let you successfully request/receive data.**

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

Python | Java | C++ | C# | VB.NET

```
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.

The Trader Workstation

Our market maker-designed IBKR 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, for an application to connect to the API there must first be a running instance of TWS or IB Gateway.

The IB Gateway

As an alternative to TWS for API users, IBKR also offers IB Gateway (IBGW). From the perspective of an API application, IB Gateway and TWS are identical; both represent a server to which an API client application can open a socket connection after the user has authenticated. With either application (TWS or IBGW), the user must manually enter their username and password into a login window. For security reasons, a headless session of TWS or IBGW without a GUI is not supported. From the user's perspective, IB Gateway may be advantageous because it is a lighter application which consumes about 40% fewer resources.

Both TWS and IBGW were designed to be restarted daily. This is necessary to perform functions such as re-downloading contract definitions in cases where contracts have been changed or new contracts have been added. Beginning in version 974+ both applications offer an autorestart feature that allows the application to restart daily without user intervention. With this option enabled, TWS or IBGW can potentially run from Sunday to Sunday without re-authenticating. After the nightly server reset on Saturday night it will be necessary to again enter security credentials.

The advantages of TWS over IBGW 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 IBGW.

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.

Connectivity

A socket connection between the API client application and TWS is established with the IBApi.EClientSocket.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 TWS on a socket port where TWS is already 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.

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:

Python | Java | C++ | C# | VB.NET

```
app.connect("127.0.0.1", args.port, clientId=0)
```

eConnect starts by requesting from the operating system that a TCP socket be opened to the specified IP address and socket port. If the socket cannot be opened, the operating system (not TWS) returns an error which is received by the API client as error code 502 to IBApi.EWrapper.error (Note: 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 it is listening for connections on a different socket port. If connecting across a network, the error can also occur if there is a firewall or antivirus program blocking connections, or if the router's IP address is not listed in the "Trusted IPs" in TWS.

After the socket has been opened, there must be an initial handshake in which information is exchanged about the highest version supported by TWS and the API. This is important because API messages can have different lengths and fields in different versions and it is necessary to have a version number to interpret received messages correctly.

- For this reason it is important that the main EReader object is not created until after a connection has been established. The initial connection results in a negotiated common version between TWS and the API client which will be needed by the EReader thread in interpreting subsequent messages.

After the highest version number which can be used for communication is established, TWS will return certain pieces of data that correspond specifically to the logged-in TWS user's session. This includes (1) the account number(s) accessible in this TWS session, (2) the next valid order identifier (ID), and (3) 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 an alternative, deprecated mode of connection used in special cases in which 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 AsyncEconnect = False.

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. 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.

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

Python

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

```
app.run()
```

infinite loop and the EWrapper call-back functions are automatically triggered.
For C#, Java, C++, and Visual Basic, we instead maintain a triple thread structure which requires the creation of a reader thread, a queue thread, and then a wrapper thread. The documentation listed here further elaborates on the structure for those languages.
Now it is time to revisit the role of IBApi.EReaderSignal initially introduced in The EClientSocket Class. 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's implementer. In the Python API, it is handled automatically by the Queue class.

Java C++ C# VB.NET

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.managedAccounts. 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::EWrapper::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.

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.

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.

Logging into multiple applications

It is not possible to login to multiple trading applications simultaneously with the same username. However, it is possible to create additional usernames for an account that can be used in different trading applications simultaneously, as long as there is not more than a single trading application logged in with a given username at a time. There are some additional cases in which it is also useful to create additional usernames:

- If TWS or IBGW is logged in with a username that is used to login to Client Portal during that session, that application will not be able to automatically reconnect to the server after the next disconnection (such as the server reset).
- A TWS or IBGW session logged into a paper trading account will not to receive market data if it is sharing data from a live user which is used to login to Client Portal.

If a different username is utilized to login to Client Portal in either of these cases, then it will not affect the TWS/IBGW session.

How to add additional usernames in Account Management

- It is important to note that market data subscriptions are setup independently for each live username.

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".

Account & Portfolio Data

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. Introducing broker (IBroker) accounts with more than 50 subaccounts or configured for on-demand account lookup cannot use reqAccountSummary with group="All". A profile name can be accepted in place of group. See Unification of Groups and Profiles.

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.

Account Summary

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.

Requesting Account Summary

Requests a specific account's summary. This method will subscribe to the account summary as presented in the TWS' Account Summary tab. Customers can specify the data received by using a specific tags value. See the Account Summary Tags section for available options.

Alternatively, many languages offer the import of AccountSummaryTags with a method to retrieve all tag values.

EClient.reqAccountSummary (

reqId: int. The unique request identifier.

group: String. 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.

tags: String. A comma separated list with the desired tags

)

Important: only **two** active summary subscriptions are allowed at a time!

Python Java C++ C# VB.NET

```
self.reqAccountSummary(9001, "All", AccountSummaryT
```

Code example:

```
from ibapi.client import *
from ibapi.wrapper import *
from ibapi.contract import Contract
import time

class TradeApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def accountSummary(self, reqId: int, account: s
        print("AccountSummary. ReqId:", reqId, "Ac

    def accountSummaryEnd(self, reqId: int):
        print("AccountSummaryEnd. ReqId:", reqId)

app = TradeApp()
app.connect("127.0.0.1", 7496, clientId=1)

time.sleep(1)

app.reqAccountSummary(9001, "All", 'NetLiquidation
app.run()
```

Account Summary 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 currency.
$LEDGER:ALL	Single flag to relay all cash balance tags" in all currencies.

Receiving Account Summary

EWrapper.accountSummary(

reqId: int. the request's unique identifier.

account: String. the account id

tag: String. the account's attribute being received.

value: String. the account's attribute's value.

currency: String. the currency on which the value is expressed.

)

Receives the account information. This method will receive the account information just as it appears in the TWS' Account Summary Window.

Python Java C++ C# VB.NET

```
def accountSummary(self, reqId: int, account: str,
  print("AccountSummary. ReqId:", reqId, "Account:
```

EWrapper.accountSummaryEnd(

reqId: String. The request's identifier.

)

Notifies when all the accounts' information has ben received. Requires TWS 967+ to receive accountSummaryEnd in linked account structures.

Python Java C++ C# VB.NET

```
def accountSummaryEnd(self, reqId: int):
    print("AccountSummaryEnd. ReqId:", reqId)
```

Cancel Account Summary

Once the subscription to account summary is no longer needed, it can be cancelled via the IBApi::EClient::cancelAccountSummary method:

EClient.cancelAccountSummary(

reqId: int. The identifier of the previously performed account request

)

Python C++

```
self.cancelAccountSummary(9001)
```

Account Updates

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. Just as with the TWS' Account Window, unless there is a position change this information is updated at a fixed interval of three minutes.
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. 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 minutes. 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.

See Profit & Loss for alternative PnL data

Requesting Account Updates

Subscribes to a 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, 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 favor of the second one.

EClient.reqAccountUpdates(

subscribe: bool. Set to true to start the subscription and to false to stop it.

acctCode: String. The account id (i.e. U123456) for which the information is requested.

)

Python Java C++ C# VB.NET

```
self.reqAccountUpdates(True, self.account)
```

Code example:

```
from ibapi.client import *
from ibapi.wrapper import *
from ibapi.contract import Contract
import time

class TradeApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def updateAccountValue(self, key: str, val: st
        print("UpdateAccountValue. Key:", key, "Va
```

```
    def updatePortfolio(self, contract: Contract, p
        print("UpdatePortfolio.", "Symbol:", contr

    def updateAccountTime(self, timeStamp: str):
        print("UpdateAccountTime. Time:", timeStamp

    def accountDownloadEnd(self, accountName: str)
        print("AccountDownloadEnd. Account:", acco

app = TradeApp()
app.connect("127.0.0.1", 7496, clientId=1)

time.sleep(1)

app.reqAccountUpdates(True, 'U123456')
app.run()
```

Receiving Account Updates

Resulting account and portfolio information will be delivered via the IBApi.EWrapper.updateAccountValue, IBApi.EWrapper.updatePortfolio, IBApi.EWrapper.updateAccountTime and IBApi.EWrapper.accountDownloadEnd

EWrapper.updateAccountValue (

key: String. The value being updated.

value: String. up-to-date value

currency: String. The currency on which the value is expressed.

accountName: String. The account identifier.

)

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.

Python Java C++ C# VB.NET

```
def updateAccountValue(self, key: str, val: str, c
    print("UpdateAccountValue. Key:", key, "Value:
```

EWrapper.updatePortfolio (

contract: Contract. The Contract for which a position is held.

position: Decimal. The number of positions held.

marketPrice: Double. The instrument's unitary price

marketValue: Double. Total market value of the instrument.

averageCost: Double. Average cost of the overall position.

unrealizedPNL: Double. Daily unrealized profit and loss on the position.

realizedPNL: Double. Daily realized profit and loss on the position.

accountName: String. Account ID for the update.

)

Receives the subscribed account's portfolio. This function will receive only the portfolio of the subscribed account. After the initial callback to updatePortfolio, callbacks only occur for positions which have changed.

Python Java C++ C# VB.NET

```
def updatePortfolio(self, contract: Contract, posi
    print("UpdatePortfolio.", "Symbol:", contract.
```

EWrapper.updateAccountTime (

timestamp: String. the last update system time.

)

Receives the last time on which the account was updated.

Python Java C++ C# VB.NET

```
def updateAccountTime(self, timeStamp: str):
    print("UpdateAccountTime. Time:", timeStamp)
```

EWrapper.accountDownloadEnd (

account: String. The account identifier.

)

Notifies when all the account's information has finished.

Python Java C++ C# VB.NET

```
def accountDownloadEnd(self, accountName: str):
    print("AccountDownloadEnd. Account:", accountN
```

Account Value Keys

When requesting reqAccountUpdates customers will receive values corresponding to various account key/value pairs. The table below documents potential responses and what they mean.

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

Key	Description
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
	Cash account: Total cash value + commodities option value – futures maintenance margin

EquityWithLoanValue-C	requirement + minimum (0, futures PNL) Margin account: Total cash value + commodities option value – futures maintenance margin requirement
EquityWithLoanValue-S	Cash account: Settled Cash Margin Account: Total cash value + stock value + bond value + (non-U.S. & Canada securities options value)
ExcessLiquidity	This value shows your margin cushion, before liquidation
ExcessLiquidity-C	Equity with Loan Value – Maintenance Margin
ExcessLiquidity-S	Net Liquidation Value – Maintenance Margin
ExchangeRate	The exchange rate of the currency to your base currency
FullAvailableFunds	Available funds of whole portfolio with no discounts or intraday credits
FullAvailableFunds-C	Net Liquidation Value – Full Initial Margin
FullAvailableFunds-S	Equity with Loan Value – Full Initial Margin
FullExcessLiquidity	Excess liquidity of whole portfolio with no discounts or intraday credits
FullExcessLiquidity-C	Net Liquidation Value – Full Maintenance Margin
FullExcessLiquidity-S	Equity with Loan Value – Full Maintenance Margin
FullInitMarginReq	Initial Margin of whole portfolio with no discounts or intraday credits
FullInitMarginReq-C	Initial Margin of commodity segment's portfolio with no discounts or intraday credits
FullInitMarginReq-S	Initial Margin of security segment's portfolio with no discounts or intraday credits
FullMaintMarginReq	Maintenance Margin of whole portfolio with no discounts or intraday credits
FullMaintMarginReq-C	Maintenance Margin of commodity segment's portfolio with no discounts or intraday credits
FullMaintMarginReq-S	Maintenance Margin of security segment's portfolio with no discounts or intraday credits
FundValue	Value of funds value (money market funds + mutual funds)
FutureOptionValue	Real-time market-to-market value of futures options
FuturesPNL	Real-time changes in futures value since last settlement
FxCashBalance	Cash balance in related IB-UKL account
GrossPositionValue	Gross Position Value in securities segment
GrossPositionValue-S	Long Stock Value + Short Stock Value + Long Option Value + Short Option Value
IndianStockHaircut	Margin rule for IB-IN accounts
InitMarginReq	Initial Margin requirement of whole portfolio
InitMarginReq-C	Initial Margin of the commodity segment in base currency
InitMarginReq-S	Initial Margin of the security segment in base currency
IssuerOptionValue	Real-time mark-to-market value of Issued Option
Leverage-S	GrossPositionValue / NetLiquidation in security segment
LookAheadNextChange	Time when look-ahead values take effect
LookAheadAvailableFunds	This value reflects your available funds at the next margin change
LookAheadAvailableFunds-C	Net Liquidation Value – look ahead Initial Margin
LookAheadAvailableFunds-S	Equity with Loan Value – look ahead Initial Margin
LookAheadExcessLiquidity	This value reflects your excess liquidity at the next margin change
LookAheadExcessLiquidity-C	Net Liquidation Value – look ahead Maintenance Margin
LookAheadExcessLiquidity-S	Equity with Loan Value – look ahead Maintenance Margin
LookAheadInitMarginReq	Initial margin requirement of whole portfolio as of next period's margin change
LookAheadInitMarginReq-C	Initial margin requirement as of next period's margin change in the base currency of the account
LookAheadInitMarginReq-S	Initial margin requirement as of next period's margin change in the base currency of the account
LookAheadMaintMarginReq	Maintenance margin requirement of whole portfolio as of next period's margin change
LookAheadMaintMarginReq-C	Maintenance margin requirement as of next period's margin change in the base currency of the account
LookAheadMaintMarginReq-S	Maintenance margin requirement as of next period's margin change in the base currency of the account
MaintMarginReq	Maintenance Margin requirement of whole portfolio
MaintMarginReq-C	Maintenance Margin for the commodity segment
MaintMarginReq-S	Maintenance Margin for the security segment
MoneyMarketFundValue	Market value of money market funds excluding mutual funds
MutualFundValue	Market value of mutual funds excluding money market funds
NetDividend	The sum of the Dividend Payable/Receivable Values for the securities and commodities segments of the account
NetLiquidation	The basis for determining the price of the assets in your account
NetLiquidation-C	Total cash value + futures PNL + commodities options value
NetLiquidation-S	Total cash value + stock value + securities options value + bond value
NetLiquidationByCurrency	Net liquidation for individual currencies
OptionMarketValue	Real-time mark-to-market value of options
PASharesValue	Personal Account shares value of whole portfolio
PASharesValue-C	Personal Account shares value in commodity segment
PASharesValue-S	Personal Account shares value in security segment
PostExpirationExcess	Total projected "at expiration" excess liquidity
PostExpirationExcess-C	Provides a projected "at expiration" excess liquidity based on the soon-to expire contracts in your portfolio in commodity segment
PostExpirationExcess-S	Provides a projected "at expiration" excess liquidity based on the soon-to expire contracts in your portfolio in security segment
PostExpirationMargin	Total projected "at expiration" margin
PostExpirationMargin-C	Provides a projected "at expiration" margin value based on the soon-to expire contracts in your portfolio in commodity segment
PostExpirationMargin-S	Provides a projected "at expiration" margin value based on the soon-to expire contracts in your portfolio in security segment
PreviousDayEquityWithLoanValue	Marginable Equity with Loan value as of 16:00 ET the previous day in securities segment
PreviousDayEquityWithLoanValue-S	IMarginable Equity with Loan value as of 16:00 ET the previous day
RealCurrency	Open positions are grouped by currency
RealizedPnL	Shows your profit on closed positions, which is the difference between your entry execution cost and exit execution costs, or (execution price + commissions to open the positions) – (execution price + commissions to close the position)
RegTEquity	Regulation T equity for universal account
RegTEquity-S	Regulation T equity for security segment
RegTMargin	Regulation T margin for universal account
RegTMargin-S	Regulation T margin for security segment
SMA	Line of credit created when the market value of securities in a Regulation T account increase in value
SMA-S	Regulation T Special Memorandum Account balance for security segment
SegmentTitle	Account segment name
StockMarketValue	Real-time mark-to-market value of stock
TBondValue	Value of treasury bonds
TBillValue	Value of treasury bills
TotalCashBalance	Total Cash Balance including Future PNL
TotalCashValue	Total cash value of stock, commodities and securities
TotalCashValue-C	CashBalance in commodity segment
TotalCashValue-S	CashBalance in security segment
TradingType-S	Account Type
UnrealizedPnL	The difference between the current market value of your open positions and the average cost, or Value – Average Cost
WarrantValue	Value of warrants
WhatIfPMEnabled	To check projected margin requirements under Portfolio Margin model

Cancel Account Updates

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.

Note: An important key passed back in IBApi.EWrapper.updateAccountValue after a call to IBApi.EClient.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")

EClient.reqAccountUpdates(

subscribe: bool. Set to true to start the subscription and to false to stop it.

acctCode: String. The account id (i.e. U123456) for

Python | Java | C++ | C#

```
self.reqAccountUpdates(False, self.account)
```

which the information is requested.

)

Account Update by Model

Requesting Account Update by Model

The IBApi.EClient.reqAccountUpdatesMulti can be used in any account structure to create simultaneous account value subscriptions from one or multiple accounts and/or models. As with IBApi.EClient.reqAccountUpdates the data returned will match that displayed within the TWS Account Window.

EClient.reqAccountUpdatesMulti (

reqId: int. Identifier to label the request

account: String. Account values can be requested for a particular account

modelCode: String. Values can also be requested for a model

ledgerAndNLV: bool. returns light-weight request; only currency positions as opposed to account values and currency positions

)

Requests account updates for account and/or model.

IBApi.EClient.reqAccountUpdatesMulti cannot be used with Account="All" in IBroker accounts with more than 50 subaccounts.

A profile name can be accepted in place of group in the account parameter for Financial Advisors

Python Java C++ C# VB.NET

```
self.reqAccountUpdatesMulti(reqId, self.account, "
```

Code example:

```
from ibapi.client import *
from ibapi.wrapper import *
import time

class TradeApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def accountUpdateMulti(self, reqId: int, accou
        print("AccountUpdateMulti. RequestId:", re

    def accountUpdateMultiEnd(self, reqId: int):
        print("AccountUpdateMultiEnd. RequestId:",

app = TradeApp()
app.connect("127.0.0.1", 7496, clientId=1)

time.sleep(1)

app.reqAccountUpdatesMulti(103, 'U123456', "", Tru

app.run()
```

Receiving Account Updates by Model

The resulting account and portfolio information will be delivered via the IBApi.EWrapper.accountUpdateMulti and IBApi.EWrapper.accountUpdateMultiEnd

EWrapper.accountUpdateMulti (

requestId: int. The id of request.

account: String. The account with updates.

modelCode: String. The model code with updates.

key: String. The name of parameter.

value: String. The value of parameter.

currency: String. The currency of parameter.
)

Provides the account updates.

EWrapper.accountUpdateMultiEnd (

requestId: int. The id of request

)

Indicates all the account updates have been transmitted.

Python Java C++ C# VB.NET

```
def accountUpdateMulti(self, reqId: int, account: s
  print("AccountUpdateMulti. RequestId:", reqId, "A
```

Python Java C++ C# VB.NET

```
def accountUpdateMultiEnd(self, reqId: int):
    print("AccountUpdateMultiEnd. RequestId:", req
```

Cancel Account Updates by Model

EClient.reqAccountUpdatesMulti (

reqId: int. Identifier to label the request

account: String. Account values can be requested for a particular account

modelCode: String. Values can also be requested for a model

ledgerAndNLV: bool. Specify false to cancel your subscription.

)

Python Java C++ C# VB.NET

```
self.reqAccountUpdatesMulti(reqId, self.account, "
```

Family Codes

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.

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.

Request Family Codes

EClient.reqFamilyCodes()

Requests family codes for an account, for instance if it is a FA, IBroker, or associated account.

Python Java C++ C# VB.NET

```
self.reqFamilyCodes()
```

Receive Family Codes

EWrapper.familyCodes(

familyCodes: FamilyCodes[]. Unique family codes array of accountIds.

)

Returns array of family codes.

Python Java C++ C# VB.NET

```
def familyCodes(self, familyCodes: ListOfFamilyCode
    print("Family Codes:", familyCode)
```

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.

Request Managed Accounts

EClient.reqManagedAccts()

Requests the accounts to which the logged user has

Python Java C++ C# VB.NET

access to.

```
self.reqManagedAccts()
```

Receive Managed Accounts

EWrapper.managedAccounts (

accountsList: String. A comma-separated string with the managed account ids.

)

Returns a string of all available accounts for the logged in user. Occurs automatically on initial API client connection.

Python Java C++ C# VB.NET

```
def managedAccounts(self, accountsList: str):
    print("Account list:", accountsList)
```

Positions

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.

Note: The reqPositions function is not available in Introducing Broker or Financial Advisor master accounts that have very large numbers of subaccounts (> 50) to optimize the performance of TWS/IB Gateway. Instead the function reqPositionsMulti can be used to subscribe to updates from individual subaccounts. Also not available with IBroker accounts configured for on-demand account lookup.

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.

Request Positions

EClient.reqPositions()

Subscribes to position updates for all accessible accounts. All positions sent initially, and then only updates as positions change.

Python Java C++ C# VB.NET

```
self.reqPositions()
```

Code example:

```
from ibapi.client import *
from ibapi.wrapper import *
import threading
import time

class TradingApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self,self)

    def position(self, account: str, contract: Cont
        print("Position.", "Account:", account, "C

    def positionEnd(self):
        print("PositionEnd")

def websocket_con():
    app.run()

app = TradingApp()
app.connect("127.0.0.1", 7496, clientId=1)

con_thread = threading.Thread(target=websocket_con,
con_thread.start()
time.sleep(1)

app.reqPositions()
time.sleep(1)
```

Receive Positions

EWrapper.position(

account: String. The account holding the position.

contract: Contract. The position's Contract

pos: decimal. The number of positions held. avgCost the average cost of the position.

avgCost: double. The total average cost of all trades for the currently held position.
)

Provides the portfolio's open positions. 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

Python Java C++ C# VB.NET

```
def position(self, account: str, contract: Contract
  print("Position.", "Account:", account, "Contract
```

Ewrapper.positionEnd()

Indicates all the positions have been transmitted. Only returned after the initial callback of EWrapper.position.

Python Java C++ C# VB.NET

```
def positionEnd(self):
  print("PositionEnd")
```

Cancel Positions Request

EClient.cancelPositions()

Cancels a previous position subscription request made with EClient.reqPositions().

Python Java C++ C# VB.NET

```
self.cancelPositions()
```

Positions By Model

The function IBApi.EClient.reqPositionsMulti can be used with any account structure to subscribe to positions updates for multiple accounts and/or models. The account and model parameters are optional if there are not multiple accounts or models available. It is more efficient to use this function for a specific subset of accounts than using IBApi.EClient.reqPositions. A profile name can be accepted in place of group in the account parameter.

Request Positions By Model

EClient.reqPositionsMulti(

requestId: int. Request's identifier.

account: String. If an account Id is provided, only the account's positions belonging to the specified model will be delivered.

modelCode: String. The code of the model's positions we are interested in.
)

Python Java C++ C# VB.NET

```
self.reqPositionsMulti(requestid, "U1234567", "")
```

Code example:

```
from ibapi.client import *
```

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.

```
from ibapi.wrapper import *
import threading
import time

class TradingApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self,self)

    def positionMulti(self, reqId: int, account: st
       print("PositionMulti. RequestId:", reqId, "A

    def positionMultiEnd(self, reqId: int):
        print("")
        print("PositionMultiEnd. RequestId:", reqId

def websocket_con():
    app.run()

app = TradingApp()
app.connect("127.0.0.1", 7497, clientId=1)

con_thread = threading.Thread(target=websocket_con,
con_thread.start()
time.sleep(1)

app.reqPositionsMulti(2, "DU1234567", "")  #To spec
time.sleep(1)

app.reqPositionsMulti(3, "Group1", "")     #To spec
time.sleep(1)
```

Receive Positions By Model

EWrapper.positionMulti(

requestId: int. The id of request

account: String. The account holding the position.

modelCode: String. The model code holding the position.

contract: Contract. The position's Contract

pos: decimal. The number of positions held.

avgCost: double. The average cost of the position.
)

Provides the portfolio's open positions.

Python | Java | C++ | C# | VB.NET

```
def positionMulti(self, reqId: int, account: str, m
  print("PositionMulti. RequestId:", reqId, "Accou
```

EWrapper.positionMultiEnd(

requestId: int. The id of request
)

Indicates all the positions have been transmitted.

Python | Java | C++ | C# | VB.NET

```
def positionMultiEnd(self, reqId: int):
      print("PositionMultiEnd. RequestId:", reqId
```

Cancel Positions By Model

EClient.cancelPositionsMulti (

requestId: int. The identifier of the request to be canceled.

)

Cancels positions request for account and/or model.

Python | Java | C++ | C# | VB.NET

```
self.cancelPositionsMulti(requestid)
```

Profit & Loss (PnL)

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 'All' subaccounts, or for a portfolio model. This is further extended to include realized P&L information at the account or individual position level.

The P&L API functions demonstrated below return the data which is displayed in the TWS Portfolio Window in current versions of TWS. As such, the P&L values are calculated based on the reset schedule specified 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.

See Account Updates for alternative PnL data.

Request P&L for individual positions

Subscribe using the IBApi::EClient::reqPnLSingle function Cannot be used with IBroker accounts configured for on-demand lookup with account = 'All'. Currently updates are returned to IBApi.EWrapper.pnlSingle approximately once per second*.

- 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.
- Introducing broker accounts without a large number of subaccounts (<50) can receive aggregate data by specifying the account as "All".
- *Cannot be used with IBroker accounts configured for on-demand lookup with account = 'All'

*subject to change in the future.

EClient.reqPnLSingle (

reqId: int. Request identifier for to track the data.

account: String. Account in which position exists

modelCode: String. Model in which position exists

conId: int. Contract ID (conId) of contract to receive daily PnL updates for. Note: does not return message if invalid conId is entered

)

Requests real time updates for daily PnL of individual positions.

Python | Java | C++ | C# | VB.NET

```
self.reqPnLSingle(requestId, "U1234567", "", 26559
```

Code example:

```
from ibapi.client import *
from ibapi.wrapper import *
import time

class TradeApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def pnlSingle(self, reqId: int, pos: Decimal, u
        print("Daily PnL Single. ReqId:", reqId, "P

app = TradeApp()
app.connect("127.0.0.1", 7496, clientId=1)

time.sleep(1)
app.reqPnLSingle(101, "U123456", "", 8314) #IBM co

app.run()
```

Receive P&L for individual positions

EWrapper.pnlSingle (

reqId: int. Request identifier used for tracking.

Python | Java | C++ | C# | VB.NET

pos: decimal. Current size of the position

dailyPnL: double. DailyPnL for the position

unrealizedPnL: double. Total unrealized PnL for the position (since inception) updating in real time

realizedPnL: double. Total realized PnL for the position (since inception) updating in real time

value: double. Current market value of the position.
)

Receives real time updates for single position daily PnL values

```
def pnlSingle(self, reqId: int, pos: Decimal, daily
  print("Daily PnL Single. ReqId:", reqId, "Positi
```

Cancel P&L request for individual positions

EClient.cancelPnLSingle (

reqId: int. Request identifier to cancel the P&L subscription for.
)

Cancels real time subscription for a positions daily PnL information.

Python Java C++ C# VB.NET

```
self.cancelPnLSingle(requestId);
```

Request P&L for accounts

Subscribe using the IBApi::EClient::reqPnL function. Updates are sent to IBApi.EWrapper.pnl.

- Introducing broker accounts with less than 50 subaccounts can receive aggregate PnL for all subaccounts by specifying 'All' as the account code.
- 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.

EClient.reqPnL (

reqId: int. Request ID to track the data.

account: String. Account for which to receive PnL updates

modelCode: String. Specify to request PnL updates for a specific model.
)

Creates subscription for real time daily PnL and unrealized PnL updates.

Python Java C++ C# VB.NET

```
self.reqPnL(reqId, "U1234567", "")
```

Code example:

```
from ibapi.client import *
from ibapi.wrapper import *
import time

class TradeApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def pnl(self, reqId: int, dailyPnL: float, unre
        print("Daily PnL. ReqId:", reqId, "DailyPnl

app = TradeApp()
app.connect("127.0.0.1", 7496, clientId=1)

time.sleep(1)
app.reqPnL(102, "U123456", "")

app.run()
```

Receive P&L for accounts

EWrapper.pnl (

reqId: int. Request identifier for tracking data.

dailyPnL: double. DailyPnL updates for the account in real time

unrealizedPnL: double. Total Unrealized PnL updates for the account in real time

realizedPnL: double. Total Realized PnL updates for the account in real time

)

Python Java C++ C# VB.NET

```
def pnl(self, reqId: int, dailyPnL: float, unrealiz
  print("Daily PnL. ReqId:", reqId, "DailyPnL:", da
```

Cancel P&L subscription requests for accounts

EClient.cancelPnL (

reqId: int. Request identifier for tracking data.
)

Cancels subscription for real time updated daily PnL params reqId

Python Java C++ C# VB.NET

```
self.cancelPnL(reqId)
```

White Branding User Info

This function will return White Branding ID ⇗ associated with the user.

Please note, that nothing will be returned if requesting username is not associated with any White Branding entity.

Requesting White Branding Info

EClient.reqUserInfo(

reqId: int. Request ID

)

Python Java C++ C# VB.NET

```
self.reqUserInfo(reqId)
```

Receiving White Branding Info

EWrapper.userInfo (

reqId: int. Identifier for the given request.

whiteBrandingId: String. Identifier for the white branded entity.
)

Python Java C++ C# VB.NET

```
def userInfo(self, reqId: int, whiteBrandingId: str
      print("UserInfo.", "ReqId:", reqId, "WhiteB
```

Bulletins

From time to time, IB sends out important News Bulletins ⇗, which can be accessed via the TWS API through the EClient.reqNewsBulletins. Bulletins are delivered via IBApi.EWrapper.updateNewsBulletin whenever there is a new bulletin. In order to stop receiving bulletins you need to cancel the subscription.

Request IB Bulletins

EClient.reqNewsBulletins (

allMessages: bool. If set to true, will return all the

Python Java C++ C# VB.NET

existing bulletins for the current day, set to false to receive only the new bulletins.
)

Subscribes to IB's News Bulletins.

```
self.reqNewsBulletins(True)
```

Receive IB Bulletins

EWrapper.updateNewsBulletin (

msgId: int. The bulletin's identifier.

msgType: int. 1: Regular news bulletin; 2: Exchange no longer available for trading; 3: Exchange is available for trading.

message: String. The news bulletin context.

origExchange: String. The exchange where the message comes from.
)

Provides IB's bulletins

Python | Java | C++ | C# | VB.NET

```
def updateNewsBulletin(self, msgId: int, msgType: 
  print("News Bulletins. MsgId:", msgId, "Type:", 
```

Cancel Bulletin Request

EClient.cancelNewsBulletin ()

Cancels IB's news bulletin subscription.

Python | Java | C++ | C# | VB.NET

```
self.cancelNewsBulletins()
```

Contracts (Financial Instruments)

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.

The Contract Object

The Contract object is an object used throughout the TWS API to define the target of your requests. Contract objects will be used for market data, portfolios, orders, executions, and even some news request. This is the staple structure used for all of the TWS API.

In all contracts, the minimum viable structure requires at least a conId and exchange, or a symbol, secType, exchange, primaryExchange, and currency. Derivatives will require additional fields, such as lastTradeDateOrExpiration, tradingClass, multiplier, strikes, and so on.

The values to the right represent the most common Contract values to pass for complete contracts. For a more comprehensive list of contract structures, please see the Contracts page.

Given additional structures for contracts are ever evolving, it is recommended to review the relevant Contract class in your programming language for a comprehensive review of what fields are available.

Contract()

ConId: int. Identifier to specify an exact contract.

Symbol: String. Ticker symbol of the underlying instrument.

SecType: String. Security type of the traded instrument.

Exchange: String. Exchange for which data or trades should be routed.

PrimaryExchange: String. Primary listing exchange of the instrument.

Currency: String. Base currency the instrument is traded on.

LastTradeDateOrContractMonth: String. For derivatives, the expiration date of the contract.

Strike: double. For derivatives, the strike price of the instrument.

Right: String. For derivatives, the right (P/C) of the instrument.

TradingClass: String. For derivatives, the trading class of the instrument.
May be used to indicate between a monthly or a weekly contract.

Contract Class Reference

Finding Contract Details in Trader Workstation

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.

Contract Details

Complete details about a contract in IB's database can be retrieved using the function IBApi.EClient.reqContractDetails. This includes information about a contract's conID, symbol, local symbol, currency, etc. which is returned in a IBApi.ContractDetails object. reqContractDetails takes as an argument a Contract object which may uniquely match one contract, 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.

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.

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.
Another function of IBApi::EClient::reqContractDetails is to request the trading schedule of an instrument via the TradingHours and LiquidHours fields. The corresponding timeZoneId field will then indicate the time zone for the trading schedule of the instrument. TWS sends these timeZoneId strings to the API from the schedule responses as-is, and may not exactly match the time zones displayed in the TWS contract description.

Possible timeZoneId values are:

- Europe/Riga
- Australia/NSW
- Europe/Warsaw
- US/Pacific
- Europe/Tallinn
- Japan
- US/Eastern
- GB-Eire
- Africa/Johannesburg
- Israel

- Europe/Vilnius
- MET
- Europe/Helsinki
- US/Central
- Europe/Budapest
- Asia/Calcutta
- Hongkong
- Europe/Moscow
- GMT

Request Contract Details

EClient.reqContractDetails (

reqId: int. Request identifier to track data.

contract: ContractDetails. the contract used as sample to query the available contracts.
Typically contains at least the Symbol, SecType, Exchange, and Currency.
)

Receives the full contract's definitions This method will return all contracts matching the requested via EClientSocket::reqContractDetails. For example, one can obtain the whole option chain with it.

Python Java C++ C# VB.NET

```
self.reqContractDetails(reqId, contract)
```

Receive Contract Details

EWrapper.contractDetails (

reqId: int. Request identifier to track data.

contract: ContractDetails. Contains the full contract object contents including all information about a specific traded instrument.
)

Receives the full contract's definitions This method will return all contracts matching the requested via EClientSocket::reqContractDetails. For example, one can obtain the whole option chain with it.

Python Java C++ C# VB.NET

```
def contractDetails(self, reqId: int, contractDeta
  print(reqId, contractDetails)
```

EWrapper.contractDetailsEnd (

reqId: int. Request identifier to track data.
)

After all contracts matching the request were returned, this method will mark the end of their reception.

Python Java C++ C# VB.NET

```
def contractDetailsEnd(self, reqId: int):
        print("ContractDetailsEnd. ReqId:", reqId)
```

Receive Bond Details

EWrapper.bondContractDetails (

reqId: int. Request identifier to track data.

contract: ContractDetails. Contains the full contract object contents including all information about a specific traded instrument.
)

Delivers the Bond contract data after this has been requested via reqContractDetails.

Python Java C++ C# VB.NET

```
def bondContractDetails(self, reqId: int, contractI
  printinstance(reqId, contractDetails)
```

Option Chains

The option chain for a given security can be returned using the function EClient.reqContractDetails. If an option contract is incompletely defined (for instance with the strike undefined) and used as an argument to EClient.reqContractDetails, a list of all matching option contracts will be returned.

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. The function EClient.reqSecDefOptParams was introduced that does not have the throttling limitation.

- It is not recommended to use EClient.reqContractDetails to receive complete option chains on an underlying, e.g. all combinations of strikes/rights/expiries.
- For very large option chains returned from EClient.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.

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.

Request Option Chains

EClient.reqSecDefOptParams (

reqId: int. The ID chosen for the request

underlyingSymbol: String. Contract symbol of the underlying.

futFopExchange: String. The exchange on which the returned options are trading. Can be set to the empty string "" for all exchanges.

underlyingSecType: String. The type of the underlying security, i.e. STK

underlyingConId: int. The contract ID of the underlying security.
)

Requests security definition option parameters for viewing a contract's option chain.

Python Java C++ C# VB.NET

```
self.reqSecDefOptParams(0, "IBM", "", "STK", 8314)
```

Receive Option Chains

EWrapper.securityDefinitionOptionParameter (

reqId: int. ID of the request initiating the callback.

underlyingConId: int. The conID of the underlying security.

tradingClass: String. The option trading class.

multiplier: String. The option multiplier.

exchange: String. Exchange for which the derivative is hosted.

expirations: HashSet. A list of the expiries for the options of this underlying on this exchange.

strikes: HashSet. A list of the possible strikes for options of this underlying on this exchange.
)

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

Python Java C++ C# VB.NET

```
def securityDefinitionOptionParameter(self, reqId:
  print("SecurityDefinitionOptionParameter.", "Req
```

Stock Symbol Search

The 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

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

Request Stock Contract Search

EClient.reqMatchingSymbols (

reqId: int. Request identifier used to track data.

pattern: String. Either start of ticker symbol or (for larger strings) company name.
)

Requests matching stock symbols.

Python | Java | C++ | C# | VB.NET

```
self.reqMatchingSymbols(reqId, "IBM")
```

Receive Searched Stock Contract

EWrapper.symbolSamples (

reqID: int. Request identifier used to track data.

contractDescription: ContractDescription[]. Provide an array of contract objects matching the requested descriptoin.
)

Returns array of sample contract descriptions

Python | Java | C++ | C# | VB.NET

```
def symbolSamples(self, reqId: int, contractDescri
        print("Symbol Samples. Request Id: ", reqI
        for contractDescription in contractDescrip
                derivSecTypes = ""
                for derivSecType in contractDescri
                        derivSecTypes += " "
                        derivSecTypes += derivSecT
                        print("Contract: conId:%s,
                                "currency:%s, deri
                                contractDescriptio
                                contractDescriptio
                                contractDescriptio
                                contractDescriptio
                                contractDescriptio
                                contractDescriptio
                                contractDescriptio
```

Error Handling

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 EWrapper.error().

- **Exceptions when no response can occur**: 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 EWrapper.error() method. The EWrapper.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".

Understanding Message Codes

The TWS uses the 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.

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.

System Message Codes

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 our servers. Your client application must pay special attention to them and handle the situation accordingly. You are very likely to lose 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.

Note:

1. 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.
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.

Error Codes

Error codes in different ranges have different indications.

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.
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, SMART_ECN, INSTINET, or PRIMEX."	This error is deprecated.
116	The order cannot be transmitted to a dead exchange.	Exchange field is invalid.
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 VWAP 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 VWAP 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.
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	Conditional submission of orders is supported for Limit, Market, MidPrice, Relative and Snap order types only. Conditional cancelation of orders is supported for Limit and MidPrice order types only.	
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
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."
200	The contract description specified for is ambiguous	Ambiguity may occur when the contract definition provided is not unique.
200		"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."
200		"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."
201	Order rejected – Reason:	An attempted order was rejected by the IB servers. See Order Placement Considerations for additional information/considerations for these errors.
202	Order cancelled – Reason:	An active order on the IB server was cancelled. See Order Placement Considerations for additional information/considerations for these errors.
203	The 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:	
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 API 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 " 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 HALTED. Please re-subscribe.	You need to re-subscribe to start receiving market depth data again.
		317	Market depth data has been RESET. Please empty deep book contents before applying any new entries.	
		319	Invalid 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.	
		322	Server error when processing an API client request.	
		323	Server error: cause – s	
		324	Server error when reading a DDE 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.	
		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 designatedLocation 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.
		355	Order size does not conform to market rule.	Check order size parameters for the specified contract from the TWS 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).	
		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.
			Volatility type if set must be 1 or 2	

		367	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.	
		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
		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 ScaleInitLevelSize
		420	Invalid real-time query.	Information about pacing violations
		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
		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.
			Delta neutral orders are only	

		429	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.AlgoStrategy 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.AlgoStrategy 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 ScalePriceAdjustInterval is not specified properly
		448	Invalid scale price adjustment interval.	ScalePriceAdjustInterval specified is invalid
		449	Unexpected scale price adjustment amount or interval.	ScalePriceAdjustValue or ScalePriceAdjustInterval specified is invalid
		481	Order size reduced.
		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.
		505	Fatal Error: Unknown message id.
		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."
		585	FA Profile is not supported anymore, use FA Group instead	"Indicates FaDataTypeEnum.PROFILES is deprecated. Use FaDataTypeEnum.GROUPS or 1 instead"
		2100	New account data requested from TWS. API client has been unsubscribed from account data.	"The TWS only allows one IBApi.EClient.reqAccountUpdates request at a time. If the client application attempts to subscribe to a second 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.reqAccountUpdates when there is an active subscription started by a different client, the TWS 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."
		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.
		2111	The Start and/or End Time for algo order BUY/SELL a contract was adjusted to use the next trading date. To modify this setting, use the Auto-adjust algo order date item on the Orders configuration page	Please go to TWS Global Configuration – "Orders" – "Settings" to correct the configuration.
		2119	Market data farm is connecting.
		2130	Warning: products are trading on the basis of currency price with factor.
		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""."
		2152	Market depth smart depth exchanges.
		2158	Sec-def data farm connection is OK	"A notification that connection to the Security definition data server is ok. This is a notification and not a true error condition, and is expected on first establishing connection."
		2168	Etrade Only Not Supported Warning	The EtradeOnly IBApi.Order attribute is no longer supported. Error received with TWS versions 983+. Remove attribute to place order.
		2169	Firm Quote Only Not Supported Warning	The firmQuoteOnly IBApi.Order attribute is no longer supported. Error received with TWS versions 983+. Remove attribute to place order.
		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
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."
10147	Order to be canceled was not found.	
10148	"OrderId that needs to be cancelled can not be cancelled, state:"	An attempt was made to cancel an order that had already been filled by the system.
10186	Requested market data is not subscribed. Delayed market data is not enabled	See Market Data Types on how to enable delayed data.
10187	Failed to request historical ticks:No market data permissions	
10189	Failed to request tick-by-tick data. Invalid Real-time Query	Trading TWS session is connected from a different IP address. Or, No market data permissions
10197	No market data during competing session	"Indicates that the user is logged into the paper account and live account simultaneously trying to request live market data using both the accounts. In such a scenario preference would be given to the live account, for more details please refer: https://ibkr.info/node/1719"
10225	"Bust event occurred, current subscription is deactivated. Please resubscribe real-time bars immediately"	
10230	"You have unsaved FA changes. Please retry 'request FA' operation later, when 'replace FA' operation is complete"	There are pending Financial Advisor configuration changes. See Financial Advisors
10231	The following Groups and/or Profiles contain invalid accounts:	"If the account(s) inside Groups or Profiles is/are incorrect in xml-formatted configuration string of replaceFA request, then the error shows list of such Groups and/or Profiles."
10233	Defaults were inherited from CASH preset during the creation of this order.	
10234	The Decision Maker field is required and not set for this order (non-desktop).	
10235	The Decision Maker field is required and not set for this order (ibbot).	
10236	Child has to be AON if parent order is AON	
10237	All or None ticket can route entire unfilled size only	
10238	Some error occured during communication with Advisor Setup web-app	
10239	This order will affect one or more accounts that are flagged because they do not fit the required risk score criteria prescribed by the group/profile/model allocation.	
10240	You must enter a valid Price Cap.	
10241	Order Quantity is expressed in monetary terms. Modification is not supported via API. Please use desktop version to revise this order.	
10242	Fractional-sized order cannot be modified via API. Please use desktop version to revise this order.	
10243	Fractional-sized order cannot be placed via API. Please use desktop version to place this order.	
10244	Cash Quantity cannot be used for this order	
10245	This financial instrument does not support fractional shares trading	
10246	This order doesn't support fractional shares trading	
10247	Only IB SmartRouting supports fractional shares	
10248	doesn't have permission to trade fractional shares	
10249	"<*****> order doesn't support fractional shares"	
10250	The size does not conform to the minimum variation of for this contract	
10251	Fractional shares are not supported for allocation orders	
10252	This non-close-position order doesn't support fractional shares trading	
10253	Clear Away orders are not supported for multi-leg combo with attached hedge.	
10254	Invalid Order: bond expired	
10268	The 'EtradeOnly' order attribute is not supported	The EtradeOnly IBApi.Order attribute is no longer supported. Error received with TWS versions 983+
10269	The 'firmQuoteOnly' order attribute is not supported	The firmQuoteOnly IBApi.Order attribute is no longer supported. Error received with TWS versions 983+
10270	The 'nbboPriceCap' order attribute is not supported	The nbboPriceCap IBApi.Order attribute is no longer supported. Error received with TWS versions 983+
10276	News feed is not allowed	The API client is not permissioned for receiving WSH news feed.
10277	News feed permissions required	The API client is not subscribed to receive WSH news feed
10278	Duplicate WSH metadata request	A request is already pending for the same API client.
10279	Failed request WSH metadata	A general error occurred when processing the request.
10280	Failed cancel WSH metadata	A general error occurred when processing the request.
10281	Duplicate WSH event data request	A request is already pending for the same API client.
10282	WSH metadata not requested	WSH metadata was not requested by first sending a reqWshMetaData request.
10283	Fail request WSH event data	A general error occurred when processing the request.
10284	Fail cancel WSH event data	A general error occurred when processing the request.
10285	Your API version does not support fractional sizing rules. Please upgrade to at least version 163	
WinError 10038	An operation was attempted on something that is not a socket.	This indicates socket connection was closed improperly. Typically for Python clients. You may refer : https://stackoverflow.com/questions/15210178/python-socket-programming-oserror-winerror-10038-an-operation-was-attempted-o

Receiving Error Messages

EWrapper.error()

reqId: int. The request identifier corresponding to the most recent reqId that maintained the error stream. This does not pertain to the orderId from placeOrder, but whatever the most recent requestId is.

errorTime: int. The Unix timestamp of when the error took place.
Note: This is only implemented for TWS API 10.33+

errorCode: int. The code identifying the error.

Python | Java | C++ | C# | VB.NET

```
def error(self, reqId: TickerId, errorTime: int, e
  print("Error. Id:", reqId, errorTime, "Code:", e
```

errorMsg: String. The error's description.

advancedOrderRejectJson: String. Advanced order reject description in json format.
)

Financial Advisors

Financial Advisors are able to manage their allocation groups from the TWS API.

Note: Modifications made through the API will effect orders placed through TWS, the TWS API, Client Portal, and the Client Portal API.

Request FA Groups and Profiles

EClient.requestFA (

faDataType: int. The configuration to change. Set to 1 or 3 as defined in the table below.
)

Requests the FA configuration as set in TWS for the given FA Group or Profile.

Python | Java | C++ | C# | VB.NET

```
self.requestFA(1)
```

requestFA FA Data Types

Type Code	Type Name	Description
1	Groups	offer traders a way to create a group of accounts and apply a single allocation method to all accounts in the group.
3	Account Aliases	let you easily identify the accounts by meaningful names rather than account numbers.

Receiving FA Groups and Profiles

EWrapper.receiveFA (

faDataType: int. Receive the faDataType value specified in the requestFA. See FA Data Types

faXmlData: String. The xml-formatted configuration.
)

Receives the Financial Advisor's configuration available in the TWS.

Python | Java | C++ | C# | VB.NET

```
def receiveFA(self, faData: FaDataType, cxml: str)
        print("Receiving FA: ", faData)
        open('log/fa.xml', 'w').write(cxml)
```

Replace FA Allocations

EClient.replaceFA (

reqId: int. Request identifier used to track data.

faDataType: int. The configuration structure to change. Set to 1 or 3 as defined above.

xml: String. XML configuration for allocation profiles or group. See Allocation Method XML Format for more details.
)

Python | Java | C++ | C# | VB.NET

```
self.replaceFa(reqId, 1, xml)
```

replaceFA FA Data Types

replaceFA Type Code	Type Name	Description
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	Account Aliases	let you easily identify the accounts by meaningful names rather than account numbers.

Note:

In order to confirm that your FA changes were saved, you may wait for the EWrapper.replaceFAEnd callback, which provides the corresponding reqId. In addition, after saving changes, it is advised to verify the new FA setup via EClient.requestFA. If it is called before changes are fully saved, you may receive an error, such as error 10230. See Message Codes.

EClient.replaceFA only accepts faDataType 1 now. Otherwise, it may trigger error 585.

EWrapper.replaceFAEnd (

reqId: int. Request identifier used to track data.

text: String. the message text.

)

Marks the ending of the replaceFA reception.

Python | Java | C++ | C# | VB.NET

```
def replaceFAEnd(self, reqId: int, text: str):
    super().replaceFAEnd(reqId, text)
    print("ReplaceFAEnd.", "ReqId:", reqId, "Text:
```

Allocation Methods and Groups

A number of methods for account allocations are available with Financial Advisor and IBroker account structures to specify how trades should be distributed across multiple accounts.

Allocation Groups can be created or modified in the Trader Workstation directly as described in TWS: Allocations and Transfers.

Alternatively, allocation groups can be created or modified through the EClient.replaceFA() method in the API.

Interactive Brokers supports two forms of allocation methods. Allocation methods that have calculations completed by Interactive Brokers, and a set of allocation methods calculated by the user and then specified.

IB-computed allocation methods

- Available Equity
- Equal Quantity
- Net Liquidation Value

User-specified allocation methods

Formerly known as Allocation Profiles

- Cash Quantity
- Percentages
- Ratios
- Shares

Allocation Method XML Format

Allocation methods for financial advisor's allocation groups are created using an XML format. The content below signifies the supported allocation groups and how to format them in their respective XML.

Available Equity

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

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfGroups>
  <Group>
    <name>MyTestProfile2</name>
    <defaultMethod>AvailableEquity</defaultMethod>
    <ListOfAccts varName="list">
      <Account>
        <acct>DU6202167</acct>
      </Account>
      <Account>
        <acct>DU6202168</acct>
      </Account>
    </ListOfAccts>
```

Client A, 200 shares to Client B, and 400 shares to Client C.

```
  </Group>
</ListOfGroups>
```

Contracts Or 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.

Example:

Assume an order for 300 shares of stock ABC is transmitted.

In the example code shown in the right side, you can see that:

1. Account A is set to receive 100.0 shares while Account B is set to receive 200.0 shares. Account A should receive 100 shares and Account B should receive 200 shares.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfGroups>
  <Group>
  <name>MyTestProfile2</name>
  <defaultMethod>ContractsOrShares</defaultMethod>

  <ListOfAccts varName="list">
  <Account>
    <acct>DU6202167</acct>
    <amount>100.0</amount>
  </Account>
  <Account>
    <acct>DU6202168</acct>
    <amount>200.0</amount>
  </Account>
  </ListOfAccts>
  </Group>
</ListOfGroups>
```

Equal Quantity

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.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfGroups>
  <Group>
    <name>MyTestProfile2</name>
    <defaultMethod>Equal</defaultMethod>
    <ListOfAccts varName="list">
      <Account>
        <acct>DU6202167</acct>
      </Account>
      <Account>
        <acct>DU6202168</acct>
      </Account>
    </ListOfAccts>
  </Group>
</ListOfGroups>
```

MonetaryAmount

The Monetary Amount method calculates the number of units to be allocated based on the monetary value assigned to each account.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfGroups>
  <Group>
  <name>MyTestProfile2</name>
  <defaultMethod>MonetaryAmount</defaultMethod>

  <ListOfAccts varName="list">
  <Account>
    <acct>DU6202167</acct>
    <amount>1000.0</amount>
  </Account>
  <Account>
    <acct>DU6202168</acct>
    <amount>2000.0</amount>
  </Account>
  </ListOfAccts>
  </Group>
</ListOfGroups>
```

Net Liquidation Value

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.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfGroups>
  <Group>
    <name>MyTestProfile2</name>
    <defaultMethod>NetLiq</defaultMethod>
    <ListOfAccts varName="list">
      <Account>
        <acct>DU6202167</acct>
      </Account>
      <Account>
        <acct>DU6202168</acct>
      </Account>
    </ListOfAccts>
  </Group>
</ListOfGroups>
```

Percentages

This method will split the total number of shares in the order between listed accounts based on the percentages you indicate.

Example:

Assume an order for 300 shares of stock ABC is transmitted.

In the example code shown in the right side, you can see that:

1. Account A is set to have 60.0 percentage while Account B is set to have 40.0 percentage. Account A should receive 180 shares and Account B should receive 120 shares.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfGroups>
  <Group>
  <name>MyTestProfile2</name>
  <defaultMethod>Percent</defaultMethod>
  <ListOfAccts varName="list">
  <Account>
    <acct>DU6202167</acct>
    <amount>60.0</amount>
  </Account>
  <Account>
    <acct>DU6202168</acct>
    <amount>40.0</amount>
  </Account>
  </ListOfAccts>
  </Group>
</ListOfGroups>
```

While making modifications to allocations for profiles, the method uses an enumerated value. The number shown below demonstrates precisely what profile corresponds to which value.

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

Note:
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.

Ratios

This method calculates the allocation of shares based on the ratios you enter.

Example:

Assume an order for 300 shares of stock ABC is transmitted.

In the example code shown in the right side, you can see that:

1. A ratio of 1.0 and 2.0 is set to Account A and Account B. Account A should receive 100 shares and Account B should receive 200 shares.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfGroups>
  <Group>
  <name>MyTestProfile2</name>
  <defaultMethod>Ratio</defaultMethod>

  <ListOfAccts varName="list">
  <Account>
    <acct>DU6202167</acct>
    <amount>1.0</amount>
  </Account>
  <Account>
    <acct>DU6202168</acct>
    <amount>2.0</amount>
  </Account>
  </ListOfAccts>
  </Group>
</ListOfGroups>
```

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, 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:

Python | Java | C++ | C# | VB.NET

```
modelOrder = Order()
modelOrder.account = "DF12345"
modelOrder.modelCode = "Technology" # model for te
self.placeOrder(self.nextOrderId(), contract, mode
```

Unification of Groups and Profiles

With TWS/IBGW build 983+, the API settings will have a new flag/checkbox, **"Use Account Groups with Allocation Methods"** (enabled by default for new users). If not enabled, groups and profiles would behave the same as before. If it is checked, group and profile functionality will be merged.

With TWS/IBGW Build 10.20+, this setting is now enabled by default, and moving forward into new versions, the two systems can be deemed as interchangeable for modifying allocation groups, placing orders, requesting account or portfolio summaries, or requesting multiple positions.

Order Placement

For advisors to place orders to their allocation groups users would simply declare their allocation group name in the order object. This would be done with the Order's faGroup field. The example to the right references a standard market order placed to our allocation group, MyTestProfile.

Python | Java | C++ | C# | VB.NET

```
order = Order()
order.action = "BUY"
order.orderType = "MKT"
order.totalQuantity = 50
order.faGroup = "MyTestProfile"
```

Market Data: Delayed

Delayed market data can **only** used with EClient.reqMktData and EClient.reqHistoricalData. This does not function for tick data.

The API can request Live, Frozen, Delayed and Delayed Frozen market data from Trader Workstation by switching market data type via the EClient.reqMarketDataType before making a market data request. A successful switch to a different (non-live) market data type for a particular market data request will be indicated by a callback to EWrapper.marketDataType with the ticker ID of the market data request which is returning a different type of data.

- A 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).
Delayed Frozen	4	Requests delayed "frozen" data for a user without market data subscriptions.

Market Data Type Behavior

1) If user sends reqMarketDataType(1) – TWS will start sending only regular (1) market data.

2) If user sends reqMarketDataType(2) – frozen, TWS will start sending regular (1) as default and frozen (2) market data. TWS sends marketDataType callback (1 or 2) indicating what market data will be sent after this callback. It can be regular or frozen.

3) If user sends reqMarketDataType(3) – delayed, TWS will start sending regular (1) as default and delayed (3) market data.

4) If user sends reqMarketDataType(4) – delayed-frozen, TWS will start sending regular (1) as default, delayed (3) and delayed-frozen (4) market data.

Interactive Brokers data will always try to provide the most up to date market data possible, but will permit additional delayed or frozen data if available upon request.

Request Market Data Type

EClient.reqMarketDataType (

marketDataType: int. Type of market data to retrieve.

)

Switches data type returned from reqMktData request to Live (1), Frozen (2), Delayed (3), or Frozen-Delayed (4).

Python | Java | C++ | C# | VB.NET

```
self.reqMarketDataType(3)
```

Receive Market Data Type

EWrapper.marketDataType (

reqId: int. Request identifier used to track data.

marketDataType: int. Type of market data to retrieve.
)

Python Java C++ C# VB.NET

```
def marketDataType(self, reqId: TickerId, marketDat
        print("MarketDataType. ReqId:", reqId, "Ty
```

Market Data: Historical

Historical Market data is available for Interactive Brokers market data subscribers in a range of methods and structures. This includes requests for historical bars, identical to the Trader Workstation, historical Time & Sales, as well as Histogram data.

Historical Data Limitations

Historical market data has it's own set of market data limitations unique to other requests such as real time market data. This section will cover all limitations that effect historical market data in the Trader Workstation API.

Historical Data Filtering

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.

As historical data at IB gets adjusted, compressed and filtered by default, there may be historical data differences if you request historical data at different time points.

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 occurs whenever one or more of the following restrictions is not observed:

Important: these 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 specialized provider.

- 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".

Unavailable Historical Data

The other historical data limitations listed are general limitations for all trading platforms:

- Bars whose 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. For example, SOXX stock moved to NASDAQ exchange on 15 Oct 2010, so no SOXX data before 15 Oct 2010 can be retrieved despite SOXX was listed in 2001. This limitation also applied to contract which specifies `SMART` as the exchange.
- Studies and indicators such as Weighted Moving Averages or Bollinger Bands are not available from the API.

Finding the Earliest Available Data Point

For many functions, such as EClient.reqHistoricalData, you will need to request market data for a contract. Given that you may not know how long a symbol has been available, you can use EClient.reqHeadTimestamp to find the first available point of data for a given whatToShow value.

ReqHeadTimeStamp counts as an ongoing historical data request, similar to using EClient.reqHistoricalData's keepUpToDate=True flag. As a result, users should always:

- Cancel timestamp requests using EClient.cancelHeadTimeStamp.
- All EClient.reqHeadTimestamp requests follow the 30 second bar limitations, regardless of which bar size value has been requested.

Requesting the Earliest Data Point

EClient.reqHeadTimestamp (

tickerId: int., A unique identifier which will serve to identify the incoming data.

contract: Contract. The IBApi.Contract you are interested in.

whatToShow: String. The type of data to retrieve. See Historical Data Types

useRTH: int. Whether (1) or not (0) to retrieve data generated only within Regular Trading Hours (RTH)

formatDate: int. Using 1 will return UTC time in YYYYMMDD-hh:mm:ss format. Using 2 will return epoch time.
)

Returns the timestamp of earliest available historical data for a contract and data type.

Python Java C++ C# VB.NET

```
self.reqHeadTimeStamp(1, ContractSamples.USStockAt
```

Receiving the Earliest Data Point

EWrapper.headTimestamp (

requestId: int. Request identifier used to track data.

headTimestamp: String. Value identifying earliest data date
)

The data requested will be returned to EWrapper.headTimeStamp.

Python Java C++ C# VB.NET

```
def headTimestamp(self, reqId, headTimestamp):
        print(reqId, headTimestamp)
```

Cancelling Timestamp Requests

EWrapper.cancelHistogramData (

tickerId: int. Request identifier used to track data.
)

A reqHeadTimeStamp request can be cancelled with

Python Java C++ C# VB.NET

```
self.cancelHeadTimeStamp(reqId)
```

EClient.cancelHeadTimestamp

Historical Bars

Historical Bar data returns a candlestick value based on the requested duration and bar size. This will always return an open, high, low, and close values. Based on which whatToShow value is used, you may also receive volume data. See the whatToShow section for more details.

Requesting Historical Bars

EClient.reqHistoricalData(

tickerId: int, A unique identifier which will serve to identify the incoming data.

contract: Contract, The IBApi.Contract object you are working with.

endDateTime: String, The request's end date and time. This should be formatted as "YYYYMMDD HH:mm:ss TMZ" or an empty string indicates current present moment).
Please be aware that endDateTime must be left as an empty string when requesting continuous futures contracts.

durationString: String, The amount of time (or Valid Duration String units) to go back from the request's given end date and time.

barSizeSetting: String, The data's granularity or Valid Bar Sizes

whatToShow: String, The type of data to retrieve.
See Historical Data Types

useRTH: bool, Whether (1) or not (0) to retrieve data generated only within Regular Trading Hours (RTH)

formatDate: bool, The format in which the incoming bars' date should be presented. Note that for day bars, only yyyyMMdd format is available.

keepUpToDate: bool, 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). If *True*, and endDateTime cannot be specified.
Supported whatToShow values: Trades, Midpoint, Bid, Ask.

mktDataOptions: TagValueList, This is a field used exclusively for internal use.

)

Python | Java | C++ | C# | VB.NET

```
self.reqHistoricalData(4102, contract, queryTime,
```

Duration

The Interactive Brokers Historical Market Data maintains a duration parameter which specifies the overall length of time that data can be collected. The duration specified will derive the bars of data that can then be collected.

Valid Duration String Units:

Unit	Description
S	Seconds
D	Day
W	Week
M	Month
Y	Year

Historical Bar Sizes

Bar sizes dictate the data returned by historical bar requests. The bar size will dictate the scale over which the OHLC/V is returned to the API.

Valid Bar Sizes:

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

Step Sizes

The functionality of market data requests are predicated on preset step sizes. As such, not all bar sizes will work with all duration values. The table listed here will discuss the smallest to largest bar size value for each duration string.

Duration Unit	Bar units allowed	Bar size Interval (Min/Max)
S	secs | mins	1 secs -> 1mins
D	secs | mins | hrs	5 secs -> 1 hours
W	sec | mins | hrs	10 secs -> 4 hrs
M	sec | mins | hrs	30 secs -> 8 hrs
Y	mins | hrs | d	1 mins-> 1 day

Max Duration Per Bar Size

The table below displays the maximum duration values allowed for a given bar.

As an example, the maximum duration for Seconds values supported for 5 seconds bars are 86400 S. This means that if I want to retrieve more than 1 day's worth of 5 second bars, I will then need to request data in increments of D (days).

Bar Size	Max Second Duration	Max Day Duration	Max Week Duration	Max Month Duration	Max Year Duration
1 secs	2000 S	(Not Supported)	(Not Supported)	(Not Supported)	(Not Supported)
5 secs	86400 S	365 D	52 W	12 M	68 Y
10 secs	86400 S	365 D	52 W	12 M	68 Y
15 secs	86400 S	365 D	52 W	12 M	68 Y
30 secs	86400 S	365 D	52 W	12 M	68 Y
1 min	86400 S	365 D	52 W	12 M	68 Y
2 mins	86400 S	365 D	52 W	12 M	68 Y
3 mins	86400 S	365 D	52 W	12 M	68 Y
5 mins	86400 S	365 D	52 W	12 M	68 Y
10 mins	86400 S	365 D	52 W	12 M	68 Y
15 mins	86400 S	365 D	52 W	12 M	68 Y
20 mins	86400 S	365 D	52 W	12 M	68 Y
30 mins	86400 S	365 D	52 W	12 M	68 Y
1 hour	86400 S	365 D	52 W	12 M	68 Y
2 hours	86400 S	365 D	52 W	12 M	68 Y
3 hours	86400 S	365 D	52 W	12 M	68 Y
4 hours	86400 S	365 D	52 W	12 M	68 Y
8 hours	86400 S	365 D	52 W	12 M	68 Y
1 day	86400 S	365 D	52 W	12 M	68 Y
1M	86400 S	365 D	52 W	12 M	68 Y
1W	86400 S	365 D	52 W	12 M	68 Y

Format Date Received

Interactive Brokers will return historical market data

Value	Description	Example

based on the format set from the request. The formatDate parameter can be provided an integer value to indicate how data should be returned.

Note: Day bars will only return dates in the yyyyMMdd format. Time data is not available.

1	String Time Zone Date	"20231019 16:11:48 America/New_York"
2	Epoch Date	1697746308
3	Day & Time Date	"1019 16:11:48 America/New_York"

Keep Up To Date

When using keepUpToDate=True for historical data requests, you will see several bars returned with the same timestamp. This is because data is updated approximately every 1-2 seconds. These updates compound until the end of the specified bar size.

Note:
keepUpToDate is only available for whatToShow: Trades, Midpoint, Bid, Ask

In our example to the below, 15 second bars are requested, and we can see the 30 second bar built out incrementally until 20231204 13:30:30 is completed. At which point, we move on to the 45th second bars. This same logic extends into minute, hourly, or daily bars.

```
Date: 20231204 13:30:30 US/Eastern, Open: 188.56, High: 188.56, Low: 188.54, Close: 188.55
Date: 20231204 13:30:30 US/Eastern, Open: 188.56, High: 188.56, Low: 188.54, Close: 188.55
Date: 20231204 13:30:30 US/Eastern, Open: 188.56, High: 188.56, Low: 188.54, Close: 188.55
Date: 20231204 13:30:30 US/Eastern, Open: 188.56, High: 188.56, Low: 188.54, Close: 188.55
Date: 20231204 13:30:30 US/Eastern, Open: 188.56, High: 188.56, Low: 188.54, Close: 188.55
Date: 20231204 13:30:30 US/Eastern, Open: 188.56, High: 188.56, Low: 188.54, Close: 188.56
Date: 20231204 13:30:30 US/Eastern, Open: 188.56, High: 188.56, Low: 188.54, Close: 188.56
Date: 20231204 13:30:30 US/Eastern, Open: 188.56, High: 188.57, Low: 188.54, Close: 188.55
Date: 20231204 13:30:45 US/Eastern, Open: 188.54, High: 188.54, Low: 188.54, Close: 188.54
```

Receiving Historical Bars

EWrapper.historicalData (

reqId: int. Request identifier used to track data.

bar: Bar. The OHLC historical data Bar. The time zone of the bar is the time zone chosen on the TWS login screen. Smallest bar size is 1 second.
)

The historical data will be delivered via the EWrapper.historicalData method in the form of candlesticks. The time zone of returned bars is the time zone chosen in TWS on the login screen.

Python Java C++ C# VB.NET

```
def historicalData(self, reqId:int, bar: BarData):
        print("HistoricalData. ReqId:", reqId, "Ba
```

Default Return Format

The text on the right is the default formatting for returning data.

The datetime value here was modified to return UTC datetime formatting.

```
Date: 20241111-16:53:00, Open: 222.97, High: 222.9
Date: 20241111-16:53:05, Open: 222.97, High: 223.0
Date: 20241111-16:53:10, Open: 223.02, High: 223.0
Date: 20241111-16:53:15, Open: 222.98, High: 222.9
```

Note: The datetime value indicates the **beginning** of the request range rather than the end. The last bar on the right would then indicate data that took place between 20241111-16:53:15 to 20241111-16:53:20.

EWrapper.historicalSchedule (

reqId: int. Request identifier used to track data.

startDateTime: String. Returns the start date and time of the historical schedule range.

endDateTime: String. Returns the end date and time of the historical schedule range.

timeZone: String. Returns the time zone referenced by the schedule.

sessions: HistoricalSession[]. Returns the full block of historical schedule data for the duration.
)

In the case of whatToShow="schedule", you will need to also define the EWrapper.historicalSchedule value. This is a unique method that will only be called in the case of the unique whatToShow value to display calendar information.

Python Java C++ C# VB.NET

```
def historicalSchedule(self, reqId: int, startDate
        print("HistoricalSchedule. ReqId:", reqId,
        for session in sessions:
                print("\tSession. Start:", session
```

EWrapper.historicalDataUpdate (

reqId: int. Request identifier used to track data.

bar: Bar. The OHLC historical data Bar. The time zone of the bar is the time zone chosen on the TWS login screen. Smallest bar size is 1 second.
)

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.

Python Java C++ C# VB.NET

```
def historicalDataUpdate(self, reqId: int, bar: Ba
        print("HistoricalDataUpdate. ReqId:", reqI
```

EWrapper.historicalDataEnd (

reqId: int. Request identifier used to track data.

start: String. Returns the starting time of the first historical data bar.

end: String. Returns the end time of the last historical data bar.
)

Marks the ending of the historical bars reception.

Python Java C++ C# VB.NET

```
def historicalDataEnd(self, reqId: int, start: str
        print("HistoricalDataEnd. ReqId:", reqId,
```

Historical Bar whatToShow

The historical bar types listed below can be used as the whatToShow value for historical bars. These values are used to request different data such as Trades, Midpoint, Bid_Ask data and more. Some bar types support more products than others. Please note the **Supported Products** section for each bar type below.

AGGTRADES

Bar Values:

Open	High	Low	Close	Volume
First traded price	Highest traded price	Lowest traded price	Last traded price	Total traded volume

Supported Products: Cryptocurrency

ASK

Bar Values:

Open	High	Low	Close	Volume
Starting ask price	Highest ask price	Lowest ask price	Last ask price	N/A

Supported Products: Bonds, CFDs, Commodities, Cryptocurrencies, ETFs, FOPs, Forex, Funds, Futures, Metals, Options, SSFs, Stocks, Structured Products, Warrants

BID

Bar Values:

Open	High	Low	Close	Volume
Starting bid price	Highest bid price	Lowest bid price	Last bid price	N/A

Supported Products: Bonds, CFDs, Commodities, Cryptocurrencies, ETFs, FOPs, Forex, Funds, Futures, Metals,

Options, SSFs, Stocks, Structured Products, Warrants

BID_ASK

Bar Values:

Open	High	Low	Close	Volume
Time average bid	Max Ask	Min Bid	Time average ask	N/A

Supported Products: Bonds, CFDs, Commodities, Cryptocurrencies, ETFs, FOPs, Forex, Funds, Futures, Metals, Options, SSFs, Stocks, Structured Products, Warrants

FEE_RATE

Bar Values:

Open	High	Low	Close	Volume
Starting Fee Rate	Highest fee rate	Lowest fee rate	Last fee rate	N/A

Supported Products: Stocks, ETFs,

HISTORICAL_VOLATILITY

Bar Values:

Open	High	Low	Close	Volume
Starting volatility	Highest volatility	Lowest volatility	Last volatility	N/A

Supported Products: ETFs, Indices, Stocks

MIDPOINT

Bar Values:

Open	High	Low	Close	Volume
Starting midpoint price	Highest midpoint price	Lowest midpoint price	Last midpoint price	N/A

Supported Products: Bonds, CFDs, Commodities, Cryptocurrencies, ETFs, FOPs, Forex, Funds, Futures, Metals, Options, SSFs, Stocks, Structured Products, Warrants

OPTION_IMPLIED_VOLATILITY

Bar Values:

Open	High	Low	Close	Volume
Starting implied volatility	Highest implied volatility	Lowest implied volatility	Last implied volatility	N/A

Supported Products: ETFs, Indices, Stocks

SCHEDULE

Bar Values:

Open	High	Low	Close	Volume
Starting ask price	Highest ask price	Lowest ask price	Last ask price	N/A

Supported Products: Bonds, CFDs, Commodities, Cryptocurrencies, ETFs, Forex, Funds, Futures, Indices, Metals, SSFs, Stocks, Structured Products, Warrants

NOTE: SCHEDULE data returns only on 1 day bars but returns historical trading schedule only with no information about OHLCV.

TRADES

Bar Values:

Open	High	Low	Close	Volume
First traded price	Highest traded price	Lowest traded price	Last traded price	Total traded volume

Supported Products: Bonds, ETFs, FOPs, Futures, Indices, Metals, Options, SSFs, Stocks, Structured Products, Warrants

NOTES: TRADES data is adjusted for splits, but not dividends.

YIELD_ASK

Bar Values:

Open	High	Low	Close	Volume
Starting ask yield	Highest ask yield	Lowest ask yield	Last ask yield	N/A

Supported Products: Indices

Note: Yield historical data only available for corporate bonds.

YIELD_BID

Bar Values:

Open	High	Low	Close	Volume
Starting bid yield	Highest bid yield	Lowest bid yield	Last bid yield	N/A

Supported Products: Indices

Note: Yield historical data only available for corporate bonds.

YIELD_BID_ASK

Bar Values:

Open	High	Low	Close	Volume
Time average bid yield	Highest ask yield	Lowest bid yield	Time average ask yield	N/A

Supported Products: Indices

Note: Yield historical data only available for corporate bonds.

YIELD_LAST

Bar Values:

Open	High	Low	Close	Volume
Starting last yield	Highest last yield	Lowest last yield	Last last yield	N/A

Supported Products: Indices

Note: Yield historical data only available for corporate bonds.

Histogram Data

Instead of returned data points as a function of time as with the function IBApi::EClient::reqHistoricalData, histograms return data as a function of price level with function IBApi::EClient::reqHistogramData

Requesting Histogram Data

EClient.reqHistogramData (

requestId: int, id of the request

contract: Contract, Contract object that is subject of query.

useRth: bool, Data from regular trading hours (1), or all available hours (0).

period: String, string value of requested date range. This will be tied to the same bar size strings as the historical bar sizes
)

Returns data histogram of specified contract.

Python Java C++ C# VB.NET

```
self.reqHistogramData(4004, contract, false, "3 da
```

Receiving Histogram Data

EWrapper.histogramData (

requestId: int. Request identifier used to track data.

data: HistogramEntry[]. Returned Tuple of histogram data, number of trades at specified price level.
)

Returns relevant histogram data.

Python Java C++ C# VB.NET

```
def histogramData(self, reqId:int, items:HistogramE
        print("HistogramData. reqId, items)
```

Cancelling Histogram Data

EClient.cancelHistogramData (

tickerId: int. Request identifier used to track data.
)

An active histogram request which has not returned data can be cancelled with EClient.cancelHistogramData

Python Java C++ C# VB.NET

```
self.reqHistogramData(4004)
```

Historical Time & Sales

The highest granularity of historical data from IB's database can be retrieved using the API function EClient.reqHistoricalTicks for historical time and sales values. Historical Time & Sales will return the same data as what is available in Trader Workstation under the Time and Sales window. This is a series of ticks indicating each trade based on the requested values.

- Historical Tick-By-Tick data is not available for combos.
- Data will not be returned from multiple trading sessions in a single request; Multiple requests must be used.
- To complete a full second, more ticks may be returned than requested.
- Time & Sales data requires a Level 1, Top Of Book market data subscription. This would be the same subscription as EClient.reqMktData() or EClient.reqHistoricalData().

Requesting Time and Sales data

EClient.reqHistoricalTicks (

requestId: *int*, id of the request

contract: *Contract*, Contract object that is subject of query.

startDateTime: *String*, i.e. "20170701 12:01:00". Uses TWS timezone specified at login.

endDateTime: *String*, i.e. "20170701 13:01:00". In TWS timezone. Exactly one of startDateTime or endDateTime must be defined.

numberOfTicks: *int*, Number of distinct data points. Max is 1000 per request.

whatToShow: *String*, (Bid_Ask, Midpoint, or Trades) Type of data requested.

useRth: *bool*, Data from regular trading hours (1), or all available hours (0).

ignoreSize: *bool*, Omit updates that reflect only changes in size, and not price. Applicable to Bid_Ask data requests.
Note: Options and Future Options will only display a value of 1, unless to indicate a removed bid/ask, which will instead return a price and size value of 0.

miscOptions: *list*, Should be defined as *null*, reserved for internal use.
)

Requests historical Time & Sales data for an instrument.

Python Java C++ C# VB.NET

```
self.reqHistoricalTicks(18001, contract, "20170712
```

Receiving Time and Sales data

Data is returned to unique functions based on what is requested in the whatToShow field.

- IBApi.EWrapper.historicalTicks for whatToShow=MIDPOINT
- IBApi.EWrapper.historicalTicksBidAsk for whatToShow=BID_ASK
- IBApi.EWrapper.historicalTicksLast for for whatToShow=TRADES

EWrapper.historicalTicks (

reqId: int, id of the request

ticks: ListOfHistoricalTick, object containing a list of tick values for the requested timeframe.

done: bool, return whether or not this is the end of the historical ticks requested.
)

For whatToShow=MIDPOINT

Python Java C++ C# VB.NET

```
def historicalTicks(self, reqId: int, ticks: ListOf
        for tick in ticks:
                print("historicalTicks. ReqId:", re
```

EWrapper.historicalTicksBidAsk (

reqId: int, id of the request

ticks: ListOfHistoricalTick, object containing a list of tick values for the requested timeframe.

done: bool, return whether or not this is the end of the historical ticks requested.
)

For whatToShow=BidAsk

Python Java C++ C# VB.NET

```
def historicalTicksBidAsk(self, reqId: int, ticks:
        for tick in ticks:
                print("historicalTicksBidAsk. ReqI
```

EWrapper.historicalTicksLast (

reqId: int, id of the request

ticks: ListOfHistoricalTick, object containing a list of tick values for the requested timeframe.

done: bool, return whether or not this is the end of the historical ticks requested.
)

For whatToShow=Last & AllLast

Python Java C++ C# VB.NET

```
def historicalTicksLast(self, reqId: int, ticks: Li
        for tick in ticks:
                print("HistoricalTickLast. ReqId:"
```

Historical Halted and Unhalted ticks

The tick attribute pastLimit is also returned with streaming Tick-By-Tick responses. Check Halted and Unhalted ticks section.

- If tick has zero price, zero size and pastLimit flag is set – this is "Halted" tick.
- If tick has zero price, zero size and followed immediately after "Halted" tick – this is "Unhalted" tick.

Historical Date Formatting

When creating dates in the TWS API, Interactive Brokers typically supports three methods:

1. Operator Time Zone
2. Exchange Time Zone
3. Coordinated Universal Time (UTC)

Operator Time Zone

Operator Time Zone is the local time set by the user in Trader Workstation. The Operator Time Zone typically maintains a unique formatting structure separate from Exchange Time Zones; however, they can match.
A user can confirm their Operator Time Zone by launching Trader Workstation then, before logging in, click "More Options >".
Users can then confirm their active Operator Time Zone by referencing the "Time Zone" field.

For US residents, this will typically appear as "America/New_York", "America/Chicago", or "America/Los_Angeles". It is essential to note the Time Zone formatting here, as "America/Chicago" is treated differently than "US/Central" as a time zone.
All requests made can now be sent as "YYYYMMDD hh:mm:ss America/Chicago" to always send dates in Central time. This same rule goes for any alternative Time Zones, such as "America/Los_Angeles" or "America/New_York".

Exchange Time Zone

The exchange Time Zone is the value the exchange itself uses to calculate time. This value is typically unique to the Operator Time Zone, but these values can overlap.
As an example, the New York Stock Exchange operates on "US/Eastern". However, the CME operates on "US/Central". This values can be programmatically requested using the EClient.reqContractDetails method, and then received from EWrapper.contractDetails in contractDetails.TimeZoneId.

Note that this will be interpreted differently from "America/Chicago".

Coordinated Universal Time (UTC)

UTC is a time standard centered around Greenwich Mean Time (GMT). UTC historical data can be formatted as "YYYYMMDD-hh:mm:ss". Please keep in mind this is based on UTC+0, and as a reference, US/Eastern time is approximately UTC-4 or UTC-5 depending on U.S. Daylight savings.

Please note GMT is unaffected by Daylight savings, and so 09:00:00 will be the same time of day year round regardless of the exchange's or your local daylight savings observation.

Modifying Returned Date

You may also log in to the Trader Workstation and modify this in the Global Configuration under API and then Settings. Here, you will find a modifiable setting labeled "Send instrument-specific attributes for dual-mode API client in" Here you can select one of the following:

- operator timezone: refers to the local timezone you have set in the Trader Workstation or IB Gateway
- instrument timezone: refers to the timezone of the requested exchange. If "SMART" is used, this will use the instrument's primary exchange.
- UTC format: refers to a standardized return using UTC as the timezone. This will be returned in the format YYYYMMDD-hh:mm:ss

Market Data: Live

Live Data Limitations

For all data, besides Delayed Watchlist Data, a paid data subscription is required to receive market data through the API. See the Market Data Subscriptions page for more information.

- Live market data and historical bars are currently not available from the API for the exchange **OSE**. Only 15 minute delayed streaming data will be available for this exchange.
- Some Available Tick Types may not be provided due to the contract details, the time that you run the code..... ,etc. To verify whether the specific Available Tick Type is provided, it is suggested to manually check the data in TWS.
- Different Available Tick Types have different updating frequency.

The bid, ask, and last size quotes are displayed in shares instead of lots.

API users have the option to configure the TWS API to work in compatibility mode for older programs, but we recommend migrating to "quotes in shares" at your earliest convenience.

To display quotes as lots, from the Global Configuration > API > Settings page, check "Bypass US Stocks market data in shares warning for API orders."

5 Second Bars

Real time and historical data functionality is combined through the 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). 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.

Request Real Time Bars

EClient.reqRealTimeBars(

tickerId: int. Request identifier used to track data.

contract: Contract. The Contract object for which the depth is being requested

barSize: int. Currently being ignored

whatToShow: String. The nature of the data being retrieved:
Available Values: TRADES, MIDPOINT, BID, ASK

useRTH: int. 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
)

Python | Java | C++ | C# | VB.NET

```
self.reqRealTimeBars(3001, contract, 5, "MIDPOINT"
```

Code example:

```
from ibapi.client import *
from ibapi.wrapper import *
from ibapi.contract import Contract
import time

class TradeApp(EWrapper, EClient):
```

realTimeBarOptions: List<TagValue>. Internal use only.

Requests real time bars.

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.

```
    def __init__(self):
        EClient.__init__(self, self)

    def realtimeBar(self, reqId: TickerId, time:int
        print("RealTimeBar. TickerId:", reqId, Rea

app = TradeApp()
app.connect("127.0.0.1", 7496, clientId=1)

contract = Contract()
contract.symbol = "AAPL"
contract.secType = "STK"
contract.currency = "USD"
contract.exchange = "SMART"

app.reqRealTimeBars(3001, contract, 5, "TRADES", 0,

app.run()
```

Receive Real Time Bars

EWrapper.realtimeBar (

reqId: int. Request identifier used to track data.

time: long. The bar's start date and time (Epoch/Unix time)

open: double. The bar's open point

high: double. The bar's high point

low: double. The bar's low point

close: double. The bar's closing point

volume: decimal. The bar's traded volume (only returned for TRADES data)

WAP: decimal. The bar's Weighted Average Price rounded to minimum increment (only available for TRADES).

count: int. The number of trades during the bar's timespan (only available for TRADES).
)

Receives the real time 5 second bars.

Python Java C++ C# VB.NET

```
def realtimeBar(self, reqId: TickerId, time:int, op
        print("RealTimeBar. TickerId:", reqId, Rea
```

Cancel Real Time Bars

EClient.cancelRealTimeBars (

tickerId: int. Request identifier used to track data.
)

Cancels Real Time Bars' subscription.

Python Java C++ C# VB.NET

```
self.cancelRealTimeBars(3001)
```

Component Exchanges

A single data request from the API can receive aggregate quotes from multiple exchanges. 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 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.

For instance, a market data request for the IBKR US contract may return the exchange mapping identifier "a6" to EWrapper.tickReqParams . Invoking the function 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.

Request Component Exchanges

EClient.reqSmartComponents (

reqId: int. Request identifier used to track data.

bboExchange: String. Mapping identifier received from EWrapper.tickReqParams
)

Returns the mapping of single letter codes to exchange names given the mapping identifier.

Python Java C++ C# VB.NET

```
self.reqSmartComponents(1018, "a6")
```

Receive Component Exchanges

EWrapper.smartComponents (

reqId: int. Request identifier used to track data.

smartComponentMap: SmartComponentMap. Unique object containing a map of all key-value pairs
)

Containing a bit number to exchange + exchange abbreviation dictionary. All IDs can be initially retrieved using reqTickParams.

Python Java C++ C# VB.NET

```
def smartComponents(self, reqId:int, smartComponent
        print("SmartComponents:")
        for smartComponent in smartComponentMap:
                print("SmartComponent.", smartComp
```

Market Depth Exchanges

To check which exchanges offer deep book data, the function EClient.reqMktDepthExchanges 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.

API 'Exchange' fields for which a market depth request would return market maker information and result in a callback to EWrapper.updateMktDepthL2 will be indicated in the results from the EWrapper.mktDepthExchanges field by a 'True' value in the 'isL2' field:

Requesting Market Depth Exchanges

EClient.reqMktDepthExchanges ()

Requests venues for which market data is returned to updateMktDepthL2 (those with market makers).

Python Java C++ C# VB.NET

```
self.reqMktDepthExchanges()
```

Receive Market Depth Exchanges

EWrapper.mktDepthExchanges (

depthMktDataDescriptions: DepthMktDataDescription[]. A list containing all

Python Java C++ C# VB.NET

available exchanges offering market depth.
)

Called when receives Depth Market Data Descriptions.

```
def mktDepthExchanges(self, depthMktDataDescriptio
        print("MktDepthExchanges:")
        for desc in depthMktDataDescriptions:
                print("DepthMktDataDescription.", 
```

Market Depth (L2)

Market depth data, also known as level II, represents an instrument's order book. Via the TWS API it is possible to obtain this information with the EClient.reqMarketDepth function. Unlike Top Market Data (Level I), 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 EClient.reqMarketDepth.

In particular, odd lot orders are not included.

It is possible to Smart-route a EClient.reqMarketDepth request to receive aggregated data from all available exchanges, similar to the TWS BookTrader display.

An integral part of processing the incoming data is monitoring EWrapper::error for message 317 "Market depth data has been RESET. Please empty deep book contents before applying any new entries." and handling it appropriately, otherwise the update process would be corrupted.

Market Depth is not support for Calendar Spreads or Combos.

Request Market Depth

Important: Please note that the languages use different method names for requesting market depth.

The C# and Visual Basic APIs use **reqMarketDepth()**.

The Python, Java, and C++ APIs use **reqMktDepth()**.

EClient.reqMarketDepth (

tickerId: int. Request identifier used to track data.

contract: Contract. The Contract for which the depth is being requested.

numRows: int. The number of rows on each side of the order book.

isSmartDepth: bool. Flag indicates that this is smart depth request.

mktDepthOptions: List. Internal use only. Leave an empty array.
)

Requests the contract's market depth (order book).

Python | Java | C++ | C# | VB.NET

```
self.reqMktDepth(2001, contract, 5, False, [])
```

Receive Market Depth

EWrapper.updateMktDepth (

tickerId: int. Request identifier used to track data.

position: int. The order book's row being updated

operation: int. Indicates a change in the row's value.:

- 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: int. 0 for ask, 1 for bid

price: double. The order's price

size: decimal. The order's size
)

Returns the order book. Used for direct routed requests only.

Python | Java | C++ | C# | VB.NET

```
def updateMktDepth(self, reqId: TickerId, position
                print("UpdateMarketDepth. ReqId:",
```

EWrapper.updateMktDepthL2 (

tickerId: int. Request identifier used to track data.

position: int. The order book's row being updated.

marketMaker: String. The exchange holding the order if isSmartDepth is True, otherwise the MPID of the market maker.

operation: int. Indicates a change in the row's value.:

- 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: int. 0 for ask, 1 for bid

price: double. The order's price

size: decimal. The order's size

isSmartDepth: bool. Flag indicating if this is smart depth response
)

Returns the order book. Used for direct routed requests only.

Python | Java | C++ | C# | VB.NET

```
def updateMktDepthL2(self, reqId: TickerId, positi
        print("UpdateMarketDepthL2. ReqId:", reqId
```

Cancel Market Depth

EClient.cancelMarketDepth (

tickerId: int. Request identifier used to track data.

isSmartDepth: bool. Flag indicates that this is smart depth request.

)

Cancel's market depth's request.

Python | Java | C++ | C# | VB.NET

```
self.cancelMktDepth(2001, False)
```

Option Greeks

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.

The implied volatility for an option given its price and the price of the underlying can be calculated with the function EClient.calculateImpliedVolatility.

Alternatively, given the price of the underlying and an implied volatility it is possible to calculate the option price using the function EClient.calculateOptionPrice.

After the request, the option specific information will be delivered via the EWrapper.tickOptionComputation method.

Request Options Greeks

EClient reqMktData (

reqId: int. Request identifier for tracking data.

contract: Contract. Contract object used for specifying an instrument.

genericTickList: String. Comma separated ids of the available generic ticks.

snapshot: bool. Set to True for snapshot data with a relevant subscription or False for live data.

regulatorySnapshot: bool. Set to True for a paid, regulatory snapshot or False for live data.

mktDataOptions: List<TagValue>. Internal use only.
)

As long as the data is requested on an Option contract, greeks will be requested automatically if underlying market data is already subscribed.

Python Java C++ C# VB.NET

```
self.reqMktData(reqId, OptionContract, "", False, 
```

Code example:

```
from ibapi.client import *
from ibapi.wrapper import *
from ibapi.contract import Contract
import threading
import time

class TradeApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def tickGeneric(self, reqId: TickerId, tickTyp
        print("TickGeneric. TickerId:", reqId, "Ti

    def tickPrice(self, reqId: TickerId, tickType:
        print(reqId, tickType, price, attrib)

    def tickSize(self, reqId: TickerId, tickType: 
        print("TickSize. TickerId:", reqId, "TickT

    def tickString(self, reqId: TickerId, tickType
        print("TickString. TickerId:", reqId, "Typ

    def tickReqParams(self, tickerId:int, minTick:
        print("TickReqParams. TickerId:", tickerId,

    def tickOptionComputation(self, reqId: TickerI
        print("TickOptionComputation. TickerId:", 

def websocket_con():
    app.run()

app = TradeApp()
app.connect("127.0.0.1", 7496, clientId=1)

con_thread = threading.Thread(target=websocket_con,
con_thread.start()

time.sleep(1)

contract = Contract()
contract.symbol = "AAPL"
contract.secType = "OPT"
contract.exchange = "SMART"
contract.currency = "USD"
contract.lastTradeDateOrContractMonth = 20240621
contract.strike = 200
contract.right = "C"
contract.multiplier = 100

app.reqMarketDataType(1)
app.reqMktData(102, contract, "", False, False, []
```

Calculating option prices

EClient calculateOptionPrice (

reqId: int. Request identifier used to track data.

contract: Contract. The Contract object for which the depth is being requested

volatility: double. Hypothetical volatility.

underPrice: double. Hypothetical option's underlying price.

optionPriceOptions: List<TagValue>. Internal use only. Send an empty tag value list.
)

Calculates an option's price based on the provided volatility and its underlying's price.

Python Java C++ C# VB.NET

```
self.calculateImpliedVolatility(5001, OptionContrac
```

Calculating historical volatility

EClient calculateImpliedVolatility (

reqId: int. Request identifier used to track data.

contract: Contract. The Contract object for which the depth is being requested

optionPrice: double. Hypothetical option price.

underPrice: double. Hypothetical option's underlying price.

impliedVolatilityOptions: List<TagValue>. Internal use only. Send an empty tag value list.
)

Calculate the volatility for an option. Request the calculation of the implied volatility based on hypothetical option and its underlying prices.

Python Java C++ C# VB.NET

```
self.calculateOptionPrice(5002, OptionContract, 0.
```

Receiving Options Data

EWrapper.tickOptionComputation (

tickerId the request's unique identifier.

field: int. 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

tickAttrib: int. 0 – return based, 1- price based.

impliedVolatility: double. the implied volatility calculated by the TWS option modeler, using the specified tick type value.

delta: double. The option delta value.

optPrice: double. The option price.

pvDividend: double. The present value of dividends expected on the option's underlying.

gamma: double. The option gamma value.

vega: double. The option vega value.

theta: double. The option theta value.

undPrice: double. The price of the underlying.
)

Receives option specific market data. This method is called when the market in an option or its underlier

Python Java C++ C# VB.NET

```
def tickOptionComputation(self, reqId: TickerId, t
        print("TickOptionComputation. TickerId:", 
```

moves. TWS's option model volatilities, prices, and deltas, along with the present value of dividends expected on that options underlier are received.

Top of Book (L1)

Streaming market data values corresponding to data shown in TWS watchlists is available via the EClient.reqMktData. 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 EClient.reqMktData, 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.

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 watchlists. 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:

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

Request Watchlist Data

EClient.reqMktData (

reqId: int. Request identifier for tracking data.

contract: Contract. Contract object used for specifying an instrument.

genericTickList: String. Comma separated ids of the available generic ticks.

snapshot: bool. Used to retrieve a single snapshot of data for those with an existing market data subscirption.

regulatorySnapshot: bool. Used to retrieve a single snapshot of paid data. Each snapshot costs $0.01.

mktDataOptions: List<TagValue>. Internal use only.
)

Requests real time market data. Returns market data for an instrument either in real time or 10-15 minutes delayed data.

Python | Java | C++ | C# | VB.NET

```
self.reqMktData(reqId, contract, "", False, False,
```

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, specific generic ticks can be requested via the genericTickList parameter of the function:

See the Available Tick Types section for more information on generic ticks.

Streaming Data Snapshots

With an exchange market data subscription, such as Network A (NYSE), Network B(ARCA), or Network C(NASDAQ) 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 EClient.reqMktData function passing in true for the snapshot parameter, the client application will receive the currently available market data once before a 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 some cases values may not be returned for all tick types.

EWrapper.tickSnapshotEnd (

tickerId: int. Request identifier used to track data.
)

When requesting market data snapshots, this market will indicate the snapshot reception is finished. Expected to occur 11 seconds after beginning of request.

Python | Java | C++ | C# | VB.NET

```
def tickSnapshotEnd(self, reqId: int):
    print("TickSnapshotEnd. TickerId:", reqId)
```

Regulatory Snapshots

The fifth argument to reqMktData specifies a regulatory snapshot request to US stocks and options.

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 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.

Requesting regulatory snapshots is subject to pacing limitations:

- No more than one request per second.

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

Receive Live Data

Note: Please be aware that in the event subsequent orders are received with the same price value, but different size values, no new tickPrice value should be returned. Only an updated tickSize will denote that a new order was retrieved with the assumption the last tickPrice value will also correlate with the new size.

EWrapper.tickGeneric (

tickerId: int. Request identifier used to track data.

field: int. The type of tick being received.

value: double. Return value corresponding to value. See Available Tick Types for more details.
)

Returns generic data back to requester. Used for an array of tick types and is used to represent general evaluations.

Python | Java | C++ | C# | VB.NET

```
def tickGeneric(self, reqId: TickerId, tickType: Ti
        print("TickGeneric. TickerId:", reqId, "Ti
```

EWrapper.tickPrice (

tickerId: int. Request identifier used to track data.

Python | Java | C++ | C# | VB.NET

field: int. The type of the price being received (i.e. ask price).

price: double. The actual price.

attribs: TickAttrib. A TickAttrib object that contains price attributes such as TickAttrib::CanAutoExecute, TickAttrib::PastLimit and TickAttrib::PreOpen.
)

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).

```
def tickPrice(self, reqId: TickerId, tickType: Tick
        print(reqId, tickType, price, attrib)
```

EWrapper.tickSize (

tickerId: int. Request identifier used to track data.

field: int. the type of size being received (i.e. bid size)

size: int. the actual size. US stocks have a multiplier of 100.
)

Market data tick size callback. Handles all size-related ticks.

Python Java C++ C# VB.NET

```
def tickSize(self, reqId: TickerId, tickType: Tick
        print("TickSize. TickerId:", reqId, "TickT
```

EWrapper.tickString (

tickerId: int. Request identifier used to track data.

field: int. The type of the tick being received

value: String. Variable containining message response.
)

Market data callback.

Note: 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.

Python Java C++ C# VB.NET

```
def tickString(self, reqId: TickerId, tickType: Ti
        print("TickString. TickerId:", reqId, "Type
```

Exchange Component Mapping

A market data request is able to return data from multiple exchanges. 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.

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. More information about Component Exchanges.

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.

EWrapper.tickReqParams (

tickerId: int. Request identifier used to track data.

minTick: Minimum tick for the contract on the exchange.

bboExchange: String. Exchange offering the best bid offer.

snapshotPermissions: Based on the snapshot parameter in EClient.reqMktData.
)

Displays the ticker with BBO exchange.

Python Java C++ C# VB.NET

```
def tickReqParams(self, tickerId:int, minTick:float
        print("TickReqParams. TickerId:", tickerId
```

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 EWrapper.rerouteMktDataReq or EWrapper.rerouteMktDepthReq respectively with details about the underlying instrument in IB's database which does have market data.

EWrapper.rerouteMktDataReq (

reqId: int. Request identifier used to track data.

conId: int. Contract identifier of the underlying instrument which has market data.

exchange: int. Primary exchange of the underlying.
)

Returns conid and exchange for CFD market data request re-route.

Python Java C++ C# VB.NET

```
def rerouteMktDataReq(self, reqId: int, conId: int,
        print("Re-route market data request. ReqId
```

EWrapper.rerouteMktDepthReq (

reqId: int. Request identifier used to track data.

conId: int. Contract identifier of the underlying instrument which has market data.

exchange: int. Primary exchange of the underlying.
)

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.

Python Java C++ C# VB.NET

```
def rerouteMktDepthReq(self, reqId: int, conId: int
        print("Re-route market depth request. ReqI
```

Cancel Watchlist Data

EClient.cancelMktData(

tickerId: int. Request identifier used to track data.
)

Cancels a watchlist market data request.

Python Java C++ C# VB.NET

```
self.cancelMktData(2001)
```

Available Tick Types

EClient.reqMktData will return data to various methods such as EWrapper.tickPrice, EWrapper.tickSize, EWrapper.tickString, etc. The values returned are dependent upon the generic tick requested and the type of data returned. The table below references what generic tick should be requested, what tick id should be returned, and to which method it will return to.

Tick Name	Description	Generic tick required	Delivery Method	Tick Id
Disable Default Market Data	Disables standard market data stream and allows the TWS & API feed to prioritize other listed generic tick types.	mdoff	–	–
Bid Size	Number of contracts or lots offered at the bid price.	–	IBApi.EWrapper.tickSize	0
Bid Price	Highest priced bid for the contract.	–	IBApi.EWrapper.tickPrice	1
Ask Price	Lowest price offer on the contract.	–	IBApi.EWrapper.tickPrice	2
Ask Size	Number of contracts or lots offered at the ask price.	–	IBApi.EWrapper.tickSize	3
Last Price	Last price at which the contract traded (does not include some trades in RTVolume).	–	IBApi.EWrapper.tickPrice	4
Last Size	Number of contracts or lots traded at the last price.	–	IBApi.EWrapper.tickSize	5
High	High price for the day.	–	IBApi.EWrapper.tickPrice	6
Low	Low price for the day.	–	IBApi.EWrapper.tickPrice	7

		Volume	Trading volume for the day for the selected contract (US Stocks: multiplier 100).	–	IBApi.EWrapper.tickSize	8
		Close Price	"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	9
		Bid Option Computation	Computed Greeks and implied volatility based on the underlying stock price and the option bid price. See Option Greeks	–	IBApi.EWrapper.tickOptionComputation	10
		Ask Option Computation	Computed Greeks and implied volatility based on the underlying stock price and the option ask price. See Option Greeks	–	IBApi.EWrapper.tickOptionComputation	11
		Last Option Computation	Computed Greeks and implied volatility based on the underlying stock price and the option last traded price. See Option Greeks	–	IBApi.EWrapper.tickOptionComputation	12
		Model Option Computation	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	13
		Open Tick	Current session's opening price. Before open will refer to previous day. The official opening price requires a market data subscription to the native exchange of the instrument.	–	IBApi.EWrapper.tickPrice	14
		Low 13 Weeks	Lowest price for the last 13 weeks. For stocks only.	165	IBApi.EWrapper.tickPrice	15
		High 13 Weeks	Highest price for the last 13 weeks. For stocks only.	165	IBApi.EWrapper.tickPrice	16
		Low 26 Weeks	Lowest price for the last 26 weeks. For stocks only.	165	IBApi.EWrapper.tickPrice	17
		High 26 Weeks	Highest price for the last 26 weeks. For stocks only.	165	IBApi.EWrapper.tickPrice	18
		Low 52 Weeks	Lowest price for the last 52 weeks. For stocks only.	165	IBApi.EWrapper.tickPrice	19
		High 52 Weeks	Highest price for the last 52 weeks. For stocks only.	165	IBApi.EWrapper.tickPrice	20
		Average Volume	The average daily trading volume over 90 days. Multiplier of 100. For stocks only.	165	IBApi.EWrapper.tickSize	21
		Open Interest	"(Deprecated not currently in use) Total number of options that are not closed."	–	IBApi.EWrapper.tickSize	22
		Option Historical Volatility	The 30-day historical volatility (currently for stocks).	104	IBApi.EWrapper.tickGeneric	23
		Option Implied Volatility	"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	IBApi.EWrapper.tickGeneric	24
		Option Bid Exchange	Not Used.	–	IBApi.EWrapper.tickString	25
		Option Ask Exchange	Not Used.	–	IBApi.EWrapper.tickString	26
		Option Call Open Interest	Call option open interest.	101	IBApi.EWrapper.tickSize	27
		Option Put Open Interest	Put option open interest.	101	IBApi.EWrapper.tickSize	28
		Option Call Volume	Call option volume for the trading day.	100	IBApi.EWrapper.tickSize	29
		Option Put Volume	Put option volume for the trading day.	100	IBApi.EWrapper.tickSize	30
		Index Future Premium	The number of points that the index is over the cash index.	162	IBApi.EWrapper.tickGeneric	31
		Bid Exchange	"For stock and options identifies the exchange(s) posting the bid price. See Component Exchanges"	–	IBApi.EWrapper.tickString	32
		Ask Exchange	"For stock and options identifies the exchange(s) posting the ask price. See Component Exchanges"	–	IBApi.EWrapper.tickString	33
		Auction Volume	The number of shares that would trade if no new orders were received and the auction were held now.	225	IBApi.EWrapper.tickSize	34
		Auction Price	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)	225	IBApi.EWrapper.tickPrice	35
		Auction Imbalance	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)	225	IBApi.EWrapper.tickSize	36
		Mark Price	"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	IBApi.EWrapper.tickPrice	37
		Bid EFP Computation	Computed EFP bid price	–	IBApi.EWrapper.tickEFP	38
		Ask EFP Computation	Computed EFP ask price	–	IBApi.EWrapper.tickEFP	39
		Last EFP Computation	Computed EFP last price	–	IBApi.EWrapper.tickEFP	40
		Open EFP Computation	Computed EFP open price	–	IBApi.EWrapper.tickEFP	41
		High EFP Computation	Computed high EFP traded price for the day	–	IBApi.EWrapper.tickEFP	42
		Low EFP Computation	Computed low EFP traded price for the day	–	IBApi.EWrapper.tickEFP	43
		Close EFP Computation	Computed closing EFP price for previous day	–	IBApi.EWrapper.tickEFP	44
		Last Timestamp	Time of the last trade (in UNIX time).	–	IBApi.EWrapper.tickString	45
		Shortable	Describes the level of difficulty with which the contract can be sold short. See Shortable	236	IBApi.EWrapper.tickGeneric	46
		RT Volume (Time & Sales)	"Last trade details (Including both ""Last"" and ""Unreportable Last"" trades). See RT Volume"	233	IBApi.EWrapper.tickString	48
		Halted	Indicates if a contract is halted. See Halted	–	IBApi.EWrapper.tickGeneric	49
		Bid Yield	Implied yield of the bond if it is purchased at the current bid.	–	IBApi.EWrapper.tickPrice	50
		Ask Yield	Implied yield of the bond if it is purchased at the current ask.	–	IBApi.EWrapper.tickPrice	51
		Last Yield	Implied yield of the bond if it is purchased at the last price.	–	IBApi.EWrapper.tickPrice	52
		Custom Option Computation	Greek values are based off a user customized price.	–	IBApi.EWrapper.tickOptionComputation	53
		Trade Count	Trade count for the day.	293	IBApi.EWrapper.tickGeneric	54
		Trade Rate	Trade count per minute.	294	IBApi.EWrapper.tickGeneric	55
		Volume Rate	Volume per minute.	295	IBApi.EWrapper.tickGeneric	56
		Last RTH Trade	Last Regular Trading Hours traded price.	318	IBApi.EWrapper.tickPrice	57
		RT Historical Volatility	30-day real time historical volatility.	411	IBApi.EWrapper.tickGeneric	58
		IB Dividends	Contract's dividends. See IB Dividends.	456	IBApi.EWrapper.tickString	59
		Bond Factor Multiplier	The bond factor is a number that indicates the ratio of the current bond principal to the original principal	460	IBApi.EWrapper.tickGeneric	60
		Regulatory Imbalance	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	IBApi.EWrapper.tickSize	61
		News	Contract's news feed.	292	IBApi.EWrapper.tickString	62
		Short-Term Volume 3 Minutes	The past three minutes volume. Interpolation may be applied. For stocks only.	595	IBApi.EWrapper.tickSize	63
		Short-Term Volume 5 Minutes	The past five minutes volume. Interpolation may be applied. For stocks only.	595	IBApi.EWrapper.tickSize	64
		Short-Term Volume 10 Minutes	The past ten minutes volume. Interpolation may be applied. For stocks only.	595	IBApi.EWrapper.tickSize	65
		Delayed Bid	Delayed bid price. See Market Data Types.	–	IBApi.EWrapper.tickPrice	66
		Delayed Ask	Delayed ask price. See Market Data Types.	–	IBApi.EWrapper.tickPrice	67
		Delayed Last	Delayed last traded price. See Market Data Types.	–	IBApi.EWrapper.tickPrice	68
		Delayed Bid Size	Delayed bid size. See Market Data Types.	–	IBApi.EWrapper.tickSize	69

Delayed Ask Size	Delayed ask size. See Market Data Types.	–	IBApi.EWrapper.tickSize	70
Delayed Last Size	Delayed last size. See Market Data Types.	–	IBApi.EWrapper.tickSize	71
Delayed High Price	Delayed highest price of the day. See Market Data Types.	–	IBApi.EWrapper.tickPrice	72
Delayed Low Price	Delayed lowest price of the day. See Market Data Types	–	IBApi.EWrapper.tickPrice	73
Delayed Volume	Delayed traded volume of the day. See Market Data Types	–	IBApi.EWrapper.tickSize	74
Delayed Close	The prior day's closing price.	–	IBApi.EWrapper.tickPrice	75
Delayed Open	Not currently available	–	IBApi.EWrapper.tickPrice	76
RT Trade Volume	"Last trade details that excludes ""Unreportable Trades"". See RT Trade Volume"	375	IBApi.EWrapper.tickString	77
Creditman mark price	Not currently available		IBApi.EWrapper.tickPrice	78
Creditman slow mark price	Slower mark price update used in system calculations	619	IBApi.EWrapper.tickPrice	79
Delayed Bid Option	Computed greeks based on delayed bid price. See Market Data Types and Option Greeks.		IBApi.EWrapper.tickPrice	80
Delayed Ask Option	Computed greeks based on delayed ask price. See Market Data Types and Option Greeks.		IBApi.EWrapper.tickPrice	81
Delayed Last Option	Computed greeks based on delayed last price. See Market Data Types and Option Greeks.		IBApi.EWrapper.tickPrice	82
Delayed Model Option	Computed Greeks and model's implied volatility based on delayed stock and option prices.		IBApi.EWrapper.tickPrice	83
Last Exchange	Exchange of last traded price		IBApi.EWrapper.tickString	84
Last Regulatory Time	Timestamp (in Unix ms time) of last trade returned with regulatory snapshot		IBApi.EWrapper.tickString	85
Futures Open Interest	Total number of outstanding futures contracts. "HSI open interest requested with generic tick 101	588	IBApi.EWrapper.tickSize	86
Average Option Volume	Average volume of the corresponding option contracts(TWS Build 970+ is required)	105	IBApi.EWrapper.tickSize	87
Delayed Last Timestamp	Delayed time of the last trade (in UNIX time) (TWS Build 970+ is required)		IBApi.EWrapper.tickString	88
Shortable Shares	Number of shares available to short (TWS Build 974+ is required)	236	IBApi.EWrapper.tickSize	89
ETF Nav Last	The last price of Net Asset Value (NAV). For ETFs: Calculation is based on prices of ETF's underlying securities. For NextShares: Value is provided by NASDAQ	577	IBApi.EWrapper.tickPrice	96
ETF Nav Frozen Last	ETF Nav Last for Frozen data	623	IBApi.EWrapper.tickPrice	97
ETF Nav High	The high price of ETF's Net Asset Value (NAV)	614	IBApi.EWrapper.tickPrice	98
ETF Nav Low	The low price of ETF's Net Asset Value (NAV)	614	IBApi.EWrapper.tickPrice	99
Estimated IPO – Midpoint	Midpoint is calculated based on IPO price range	586	IBApi.EWrapper.tickGeneric	101
Final IPO Price	Final price for IPO	586	IBApi.EWrapper.tickGeneric	102
Delayed Yield Bid	Delayed implied yield of the bond if it is purchased at the current bid.	–	IBApi.EWrapper.tickPrice	103
Delayed Yield Ask	Delayed implied yield of the bond if it is purchased at the current ask.	–	IBApi.EWrapper.tickPrice	104

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.

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.

Receiving the actual number of shares available to short requires TWS 974+. For detailed information about shortability data (shortable shares, fee rate) available outside of TWS, IB also provides an FTP site. For more information on the FTP site, see knowledge base article 2024

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.

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 setting in TWS 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.

Note: Please be aware that RT Volume is not supported with Cryptocurrencies.

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.

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.

Tick By Tick Data

Tick-by-tick data has been available since TWS v969 and API v973.04.

In TWS, tick-by-tick data is available in the Time & Sales Window.

From the API, this corresponds to the function EClient.reqTickByTickData. The maximum number of simultaneous tick-by-tick subscriptions allowed for a user is determined by the same formula used to calculate maximum number of market depth subscriptions Limitations. For some securities, getting tick-by-tick data requires Level 2 data bundles.

- Real time tick-by-tick data is currently not available for options. Historical tick-by-tick data is available.
- 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 for indices is only provided for indices which are on CME.
- Tick-by-tick data is not available for combos.
- No more than 1 tick-by-tick request can be made for the same instrument within 15 seconds.
- Time & Sales data requires a Level 1, Top Of Book market data subscription. This would be the same subscription as EClient.reqMktData() or EClient.reqHistoricalData().

Request Tick By Tick Data

EClient.reqTickByTickData (

reqId: int. unique identifier of the request.

contract: Contract. the contract for which tick-by-tick data is requested.

tickType: String. tick-by-tick data type: "Last", "AllLast", "BidAsk" or "MidPoint".

numberOfTicks: int. If a non-zero value is entered, then historical tick data is first returned via one of the Historical Time and Sales Ewrapper Methods respectively. (Max number of historical Ticks is 1000)

ignoreSize: bool. Omit updates that reflect only changes in size, and not price. *Applicable to Bid_Ask data requests.*
)

Requests tick by tick or Time & Sales data.

Note:

EClient.reqTickByTickData() uses Max Market Depth (Level II) data lines, instead of market data lines (Level I). For market data lines, please check: https://www.interactivebrokers.com/campus/ibkr-api-page/market-data-subscriptions/#market-data-lines

Python Java C++ C# VB.NET

```
self.reqTickByTickData(19001, contract, "Last", 0,
```

Receive Tick By Tick Data

EWrapper.tickByTickAllLast (

reqId: int. unique identifier of the request.

tickType: int. 0: "Last" or 1: "AllLast".

time: long. tick-by-tick real-time tick timestamp.

price: double. tick-by-tick real-time tick last price.

size: decimal. tick-by-tick real-time tick last size.

tickAttribLast: TickAttribLast. tick-by-tick real-time last tick attribs (bit 0 – past limit, bit 1 – unreported).

exchange: String. tick-by-tick real-time tick exchange.

specialConditions: String. tick-by-tick real-time tick special conditions. Conditions under which the operation took place (Refer to Trade Conditions Page)
)

Returns "Last" or "AllLast" tick-by-tick real-time tick.

Python Java C++ C# VB.NET

```
def tickByTickAllLast(self, reqId: int, tickType: 
        print(" ReqId:", reqId, "Time:", time, "Pr
```

EWrapper.tickByTickBidAsk (

reqId: int. unique identifier of the request.

time: long. timestamp of the tick.

bidPrice: double. bid price of the tick.

askPrice: double. ask price of the tick.

bidSize: decimal. bid size of the tick.

askSize: decimal. ask size of the tick.

tickAttribBidAsk: TickAttribBidAsk. tick-by-tick real-time bid/ask tick attribs (bit 0 – bid past low, bit 1 – ask past high).
)

Returns "BidAsk" tick-by-tick real-time tick.

Python Java C++ C# VB.NET

```
def tickByTickBidAsk(self, reqId: int, time: int,
        print("BidAsk. ReqId:", reqId, "Time:", ti
```

EWrapper.tickByTickMidPoint (

reqId: int. Request identifier used to track data.

time: long. Timestamp of the tick.

midPoint: double. Mid point value of the tick.
)

Returns "MidPoint" tick-by-tick real-time tick.

Python Java C++ C# VB.NET

```
def tickByTickMidPoint(self, reqId: int, time: int,
        print("Midpoint. ReqId:", reqId, "Time:",
```

Cancel Tick By Tick Data

EClient.cancelTickByTickData (

requestId: int. Request identifier used to track data.
)

Cancels specified tick-by-tick data.

Python Java C++ C# VB.NET

```
self.cancelTickByTickData(19001)
```

Halted and Unhalted ticks

The Tick-By-Tick attribute has been introduced. The tick attribute *pastLimit* is also returned with historical Tick-By-Tick responses.

- If tick has zero price, zero size and pastLimit flag is set – this is "Halted" tick.
- If tick has zero price, zero size and followed immediately after "Halted" tick – this is "Unhalted" tick.

Market Scanner

Some scans in the TWS Advanced Market Scanner can be accessed via the TWS API through the EClient.reqScannerSubscription.

Results are delivered via EWrapper.scannerData and the 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.

Since the EClient.reqScannerSubscription request keeps a subscription open you will keep receiving periodic updates until the request is cancelled via EClient.cancelScannerSubscription :

Scans are limited to a maximum result of 50 results per scan code, and only 10 API scans can be active at a time. scannerSubscriptionFilterOptions has been added to the API to allow for generic filters. This field is entered as a list of TagValues which have a tag followed by its value, e.g. TagValue("usdMarketCapAbove", "10000") indicates a market cap above 10000 USD. Available filters can be found using the EClient.reqScannerParameters function.

A string containing all available XML-formatted parameters will then be returned via EWrapper.scannerParameters.

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 ⇗.

Market Scanner Parameters

A string containing all available XML-formatted parameters will then be returned via EWrapper.scannerParameters.

Request Market Scanner Parameters

EClient.reqScannerParameters ()

Requests an XML list of scanner parameters valid in TWS.

Python Java C++ C# VB.NET

```
self.reqScannerParameters()
```

Receive Market Scanner Parameters

EWrapper.scannerParameters (

xml: String. The xml-formatted string with the available parameters.
)

Provides the xml-formatted parameters available from TWS market scanners (not all available in API).

Python Java C++ C# VB.NET

```
def scannerParameters(self, xml: str):
        open('log/scanner.xml', 'w').write(xml)
        print("ScannerParameters received.")
```

Market Scanner Subscription

All values used for the ScannerSubscription object are pulled from EClient.scannerParams response. The XML tree will relay a tree containing a corresponding code to each ScannerSubscription field as documented below.

instrument: `<ScanParameterResponse> <InstrumentList> <Instrument> <type>`

Location Code: `<ScanParameterResponse> <LocationTree> <Location> <LocationTree> <Location> <locationCode>`

Scan Code: `<ScanParameterResponse> <ScanTypeList> <ScanType> <scanCode>`

Subscription Options should be an empty array of TagValues.

Filter Options: `<ScanParameterResponse> <FilterList> <RangeFilter> <AbstractField> <code>`

ScannerSubscription()

Instrument: String. Instrument Type to use.

Location Code: String. Country or region for scanner to search.

Scan Code: String. Value for scanner to sort by.

Subscription Options: Array of TagValues. For internal use only.

Filter Options: Array of TagValues. Contains an array of TagValue objects which filters the scanner subscription.

Request Market Scanner Subscription

EClient.reqScannerSubscription (

reqId: int. Request identifier used for tracking data.

subscription: ScannerSubscription. Object containing details on what values should be used to construct and sort the list.

scannerSubscriptionOptions: List. Internal use only.

scannerSubscriptionFilterOptions: List. List of values used to filter the results of the scanner subscription. May result in an empty scanner response from over-filtering.
)

Starts a subscription to market scan results based on the provided parameters.

Python Java C++ C# VB.NET

```
self.reqScannerSubscription(7002, scannerSubscript
```

Receive Market Scanner Subscription

EWrapper.scannerData (

reqId: int. Request identifier used to track data.

rank: int. The ranking position of the contract in the scanner sort.

contractDetails: ContractDetails. Contract object of the resulting object.

distance: String. Internal use only.

benchmark: String. Internal use only.

projection: String. Internal use only.

legStr: String. Describes the combo legs when the scanner is returning EFP
)

Provides the data resulting from the market scanner request.

Python Java C++ C# VB.NET

```
def scannerData(self, reqId: int, rank: int, contr
        print("ScannerData. ReqId:", reqId, ScanDa
```

Cancel Market Scanner Subscription

EClient.cancelScannerSubscription (

tickerId: int. Request identifier used to track data.
)

Cancels the specified scanner subscription using the tickerId.

Python Java C++ C# VB.NET

```
self.cancelScannerSubscription(7003)
```

Order Management

Commission Report

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 objects. This allows to obtain the full picture of the order's execution and the resulting commissions.

- Advisors executing allocation orders will receive execution details and commissions for the allocation order itself. To receive allocation details and commissions for a specific subaccount IBApi.EClient.reqExecutions can be used.

EWrapper.commissionReport (

commissionReport: CommissionReport. Returns a commissions report object containing the fields execId,

Python Java C++ C# VB.NET

commission, currency, realizedPnl, yield, and yieldRedemptionDate.
)

Provides the CommissionReport of an Execution

```
def commissionReport(self, commissionReport: Commi
    print("CommissionReport.", commissionReport)
```

Execution Details

IBApi.Execution and IBApi.CommissionReport can be requested on demand via the IBApi.EClient.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.

Once all matching executions have been delivered, an IBApi.EWrapper.execDetailsEnd event will be triggered.

Important: By default, only those executions occurring since midnight for that particular account will be delivered. If you want to request executions up to last 7 days, TWS's Trade Log setting "Show trades for ..." must be adjusted to your requirement. Please note, that IB Gateway would be unable to change the Trade Log's settings, thus limited to only executions since midnight to be delivered.

ExecID Behavior

If a correction to an execution is published it will be received as an additional IBApi.EWrapper.execDetails 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.

By default, most ExecID values will return as 4-segment alphanumeric sequence to identify each unique order. In the case of Combo orders, you may encounter a 5-segment alphanumeric sequence which will be used to denote per-leg executions. As an example, if a 1:1 combo for 200 shares of both contracts is placed, the first leg may fill for 200 shares, then leg 2 may fill for 100 in one execution, and then another execution for leg 2 of 100. The fifth segment will distinguish between these unique inner-combo executions.

The Execution Object

The Execution object is used to maintain all data related to a user's traded orders. This can be used in both querying execution details and navigating received data. The details provided will display all information pertaining to the execution, including how many shares were filled, the price of the execution, and what time it took place.

Execution()

OrderId: int. The API client's order Id. May not be unique to an account.

ClientId: int. The API client identifier which placed the order which originated this execution.

ExecId: String. The execution's identifier. Each partial fill has a separate ExecId. A correction is indicated by an ExecId which differs from a previous ExecId in only the digits after the final period, e.g. an ExecId ending in ".02" would be a correction of a previous execution with an ExecId ending in ".01".

Time: String. The execution's server time.

AcctNumber: String. The account to which the order was allocated.

Exchange: String. The exchange where the execution took place.

Side: String. Specifies if the transaction was buy or sale BOT for bought, SLD for sold.

Shares: decimal. The number of shares filled.

Price: double. The order's execution price excluding commissions.

PermId: int. The TWS order identifier. The PermId can be 0 for trades originating outside IB.

Liquidation: int. Identifies whether an execution occurred because of an IB-initiated liquidation.

CumQty: decimal. Cumulative quantity. Used in regular trades, combo trades and legs of the combo.

AvgPrice: double. Average price. Used in regular trades, combo trades and legs of the combo. Does not include commissions.

OrderRef: String. 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.

EvRule: String. 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.

EvMultiplier: double. 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.

ModelCode: String. model code

LastLiquidity: Liquidity. The liquidity type of the execution.

pendingPriceRevision: bool. Describes if the execution is still pending price revision.

Given additional structures for executions are ever evolving, it is recommended to review the relevant Execution class in your programming language for a comprehensive review of what fields are available.

Execution Class Reference

Request Execution Details

EClient.reqExecutions (

reqId: int. The request's unique identifier.

filter: ExecutionFilter. The filter criteria used to determine which execution reports are returned.
)

Requests current day's (since midnight) executions and commission report matching the filter. Only the current day's executions can be retrieved.

Python Java C++ C# VB.NET

```
self.reqExecutions(10001, ExecutionFilter())
```

Receive Execution Details

EWrapper.execDetails (

reqId: int. The request's identifier.

contract: Contract. The Contract of the Order.

execution: Execution. The Execution details.
)

Provides the executions which happened in the last 24 hours.

Python Java C++ C# VB.NET

```
def execDetails(self, reqId: int, contract: Contra
  print("ExecDetails. ReqId:", reqId, "Symbol:", co
```

EWrapper.execDetailsEnd (

reqId: int. The request's identifier
)

Indicates the end of the Execution reception.

Python Java C++ C# VB.NET

```
def execDetailsEnd(self, reqId: int):
        print("ExecDetailsEnd. ReqId:", reqId)
```

Open Orders

EWrapper.openOrder (

orderId: int. The order's unique id

contract: Contract. The order's Contract.

order: Order. The currently active Order.

orderState: OrderState. The order's OrderState
)

Feeds in currently open orders.

EWrapper.openOrderEnd ()

Notifies the end of the open orders' reception.

Python Java C++ C# VB.NET

```
def openOrder(self, orderId: OrderId, contract: Co
    print(orderId, contract, order, orderState)
```

Python Java C++ C# VB.NET

```
def openOrderEnd(self):
        print("OpenOrderEnd")
```

Order Status

EWrapper.orderStatus (

orderId: int. The order's client id.

status: String. The current status of the order.

filled: decimal. Number of filled positions.

remaining: decimal. The remnant positions.

avgFillPrice: double. Average filling price.

permId: int. The order's permId used by the TWS to identify orders.

parentId: int. Parent's id. Used for bracket and auto trailing stop orders.

lastFillPrice: double. Price at which the last positions were filled.

clientId: int. API client which submitted the order.

whyHeld: String. 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: double. If an order has been capped, this indicates the current capped price.
)

Gives the up-to-date information of an order every time it changes. Often there are duplicate orderStatus messages.

Python Java C++ C# VB.NET

```
def orderStatus(self, orderId: OrderId, status: st
        super().orderStatus(orderId, status, fille
```

Understanding Order Status Message

Status Code	Description
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.

Requesting 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.

API client's orders

The IBApi.EClient.reqOpenOrders method allows to obtain all active orders submitted by the client application 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.

When an order is bound by API client 0 there will be callback to IBApi::EWrapper::orderBound. This indicates the mapping between API order ID and permID. The IBApi.EWrapper.orderBound callback in response to newly bound orders that indicates the mapping between the permID (unique account-wide) and API Order ID (specific to an API client). 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.

EClient.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.

Python Java C++ C# VB.NET

```
self.reqOpenOrders()
```

All submitted orders

EClient.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.

Python Java C++ C# VB.NET

```
self.reqAllOpenOrders()
```

Manually Submitted TWS Orders

EClient.reqAutoOpenOrders (

autoBind: bool. 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.
)

Requests status updates about future orders placed from TWS. Can only be used with client ID 0.

Important: only those applications connecting with client Id 0 will be able to take over manually submitted orders

Python Java C++ C# VB.NET

```
self.reqAutoOpenOrders(True)
```

Order Binding Notification

EWrapper.orderBound (

orderId: long. IBKR permId.

apiClientId: int. API client id.

apiOrderId: int. API order id.
)

Response to API bind order control message.

Python | Java | C++ | C# | VB.NET

```
def orderBound(self, orderId: int, apiClientId: int
        print("OrderBound.", "OrderId:", intMaxStr
```

Retrieving Completed Orders

EClient.reqCompletedOrders allows users to request all orders for the given day that are no longer modifiable. This will include orders have that executed, been rejected, or have been cancelled by the user. Clients may use these requests in order to retain a roster of those order submissions that are no longer traceable via reqOpenOrders.

Requesting Completed Orders

EClient.reqCompletedOrders(

apiOnly: bool.
Determines if only API orders should be returned or if TWS submitted orders should be included.

)

Python | Java | C++ | C# | VB.NET

```
self.reqCompletedOrders(True)
```

Receiving Completed Orders

EWrapper.completedOrders(

contract: Contract. The order's Contract.
order: Order. The currently active Order.
orderState: OrderState. The order's OrderState
)

Python | Java | C++ | C# | VB.NET

```
def completedOrder(self, orderId: OrderId, contract
    print(orderId, contract, order, orderState)
```

Orders

The Order Object

The order object is an essential piece of the TWS API which is used to both place and manage orders. This is primarily built with an ever increasing range of attributes used to create the best order possible. With that being said, the value to the right represents the required fields in order to place or reference any order. Keep in mind that there are several other attributes that can and should be referenced.

Given additional structures for orders are ever evolving, it is recommended to review the relevant order class in your programming language for a comprehensive review of what fields are available.

Order()

action: String. Determines whether the contract should be a BUY or SELL.

auxPrice: double. Used to determine the stop price for STP, STP LMT, and TRAIL orders.

lmtPrice: double. Used to determine the limit price for LMT, STP LMT, and TRAIL orders.

orderType: String. Specify the type of order to place. For example, MKT, LMT, STP.

tif: String. Time in force for the order. Default tif is DAY.

totalQuantity: decimal. Total size of the order.

Order Class Reference

Exercise Options

Options are exercised or lapsed from the API with the function 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

EClient.exerciseOptions (

tickerId: int. Exercise request's identifier

contract: Contract. the option Contract to be exercised.

exerciseAction: int. Set to 1 to exercise the option, set to 2 to let the option lapse.

exerciseQuantity: int. Number of contracts to be exercised

account: String. Destination account

ovrd: int. Specifies whether your setting will override the system's natural action.
Set to 1 to override, set to 0 not to.

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.

manualOrderTime: String. Specify the time at which the options should be exercised. An empty string will assume the current time.
Required TWS API 10.26 or higher.
)

Exercises an options contract.

Note: this function is affected by a TWS setting which specifies if an exercise request must be finalized.

Python | Java | C++ | C# | VB.NET

```
self.exerciseOptions(5003, contract, 1, 1, self.ac
```

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 EClient.reqMarketRule.

The function EClient.reqContractDetails when used with a Contract object will return contractDetails object to the contractDetails 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 EClient.reqMarketRule. The rule is returned to the function EWrapper.marketRule.

- 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

Request Market Rule

EClient.reqMarketRule (

marketRuleId: int. The id of market rule
)

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 EClient.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.

Python Java C++ C# VB.NET

```
self.reqMarketRule(26)
```

Receive Market Rule

EWrapper.marketRule (

marketRuleId: int. Market Rule ID requested.

priceIncrements: PriceIncrement[]. Returns the available price increments based on the market rule.
)

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

Python Java C++ C# VB.NET

```
def marketRule(self, marketRuleId: int, priceIncre
        print("Market Rule ID: ", marketRuleId)
        for priceIncrement in priceIncrements:
        print("Price Increment.", priceIncrement)
```

Place Order

Orders are submitted via the EClient.placeOrder method. From the snippet below, note how a variable holding the nextValidId is incremented automatically.

Immediately after an order is submitted correctly, the TWS will start sending events concerning the order's activity via EWrapper.openOrder and EWrapper.orderStatus

Advisors executing allocation orders will receive execution details and commissions for the allocation order itself. To receive allocation details and commissions for a specific subaccount EClient.reqExecutions can be used.

An order can be sent to TWS but not transmitted to the IB server by setting the 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.

EClient.placeOrder (

id: int. The order's unique identifier. 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: Contract. The order's contract

order: Order. The order object.
)

Places or modifies an order.

Python Java C++ C# VB.NET

```
self.placeOrder(orderId, contract, order)
```

Order Placement Considerations

When placing orders via the API and building a robust trading system, it is important to monitor for callback notifications, specifically
for IBApi::EWrapper::error, IBApi::EWrapper::orderStatus changes, IBApi::EWrapper::openOrder warnings, and IBApi::EWrapper::execDetails to ensure proper operation.

If you experience issues with orders you place via the API, such as orders not filling, the first thing to check is what these callbacks returned. Your order may have been rejected or cancelled. If needed, see the API Log section, for information on obtaining your API logs or submitting them for review.

Common cases of order rejections, cancellations, and warnings, and the corresponding message returned:

- If an order is subject to a large size (LGSZ) reject, the API client would receive **Error (201)** via IBApi::EWrapper::error. The error text would indicate that order size too large and suggest another smaller size.
  - In accordance with our regulatory obligations as a broker, we cannot accept Large Limit Orders for #### shares of ABCD that you have submitted. Please submit a smaller order (not exceeding ###) *or convert your order to an algorithmic Order {IBALGO} {conditional on instrument}*
- If an order is subject to price checks the client may receive status (cancelled) + **Error (202)** via IBApi.EWrapper.orderStatus and IBApi::EWrapper::error. The error text would indicate the price is too far from current price.
  - In accordance with our regulatory obligations as a broker, we cannot accept your order at the limit price ### you selected because it is too far through the market. Please submit your order using a limit price that is closer to the current market price ###
- The client may receive warning Text via IBApi::EWrapper::openOrder indicating that the order could be subject to price capping.
  - If your order does not immediately execute, in accordance with our regulatory obligations as a broker we may, depending on market conditions, reject your order if the limit price of your order is more than allowed distance from the current reference price. This is designed to ensure that the price of your order is in line with an orderly market and reduce the impact your order has on the market. Please note that such rejection will result in you not receiving a fill.
  - *mktCapPrice* – If an order has been capped, this indicates the current capped price (returned to IBApi.EWrapper.orderStatus)

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 EClient.placeOrder can then be called with the same fields as the open order, except for the parameter to modify. This includes the Order.OrderId, which must match the 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.
- 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.

Cancelling an Order

An order can be cancelled from the API with the functions EClient.cancelOrder and EClient::reqGlobalCancel.

EClient.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).

EClient.reqGlobalCancel will cancel all open orders, regardless of how they were originally placed.

Cancel Individual Order

EClient.cancelOrder (

orderId: int. Specify which order should be cancelled by its identifier.

Python Java C++ C# VB.NET

orderCancel: orderCancel. An OrderCancel object that can receive the manualOrderCancelTime, manualOrderIndicator, and extOperator fields. See OrderCancel Reference for more insight on the OrderCancel class.
)

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.

```
self.cancelOrder(orderId, "")
```

Cancel All Open Orders

EClient.reqGlobalCancel ()

This method will cancel ALL open orders including those placed directly from TWS.

Python | Java | C++ | C# | VB.NET

```
self.reqGlobalCancel()
```

Test Order Impact (WhatIf)

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.OrderWhatIf 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, 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 in the EWrapper.openOrder callback..

This is equivalent to creating a order ticket in TWS, clicking "Preview", and viewing the information in the "Margin Impact" panel.

Trigger Methods

The Trigger Method defined in the IBApi.Order class specifies how simulated stop, stop-limit, and trailling stops, and conditional orders are triggered. Valid values are:

- 0 – The default method for instrument
- 1 – "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

Below is a table which indicates whether a given secType is compatible with bid/ask-driven or last-driven trigger methods (method 7 only used in iBot alerts)

secType	Bid/Ask-driven (1, 4, 8)	Last-driven (2, 3)	Default behavior	Notes
STK	yes	yes	Last	The double bid/ask is used for OTC stocks
CFD	yes	yes	Last	
CFD – Index	yes	n/a	n/a	Ex IBUS500
OPT	yes	yes	US OPT: Double bid/ask; Other: Last	
FOP	yes	yes	Last	
WAR	yes	yes	Last	
IOPT	yes	yes	Last	
FUT	yes	yes	Last	
COMBO	yes	yes	Last	
CASH	yes	n/a	Bid/ask	
CMDTY	yes	n/a	Last	
IND	n/a	yes	n/a	For conditions only

Important notes :

- If an incompatible triggerMethod and secType are used in your API order, the order may never trigger.
- These trigger methods only apply to stop orders simulated by IB. If a stop-variant is handled natively, the trigger method specified is ignored. See our Stop Orders page for more information.

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. 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. 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.

News

API news requires news subscriptions that are specific to the API; most news services in TWS are not also available in the API. There are three API news services 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 (BRF)
- Benzinga Pro (BZ)
- Fly on the Wall (FLY)

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.

News Providers

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. A list of available subscribed news sources is returned to the function IBApi::EWrapper::newsProviders

Request News Providers

EClient.reqNewsProviders()

Requests news providers which the user has subscribed to.

Python | Java | C++ | C# | VB.NET

```
self.reqNewsProviders()
```

Receive News Providers

EWrapper.newsProviders (

newsProviders: NewsProviders[]. Unique array containing all available news sources.
)

Returns array of subscribed API news providers for this user

Python | Java | C++ | C# | VB.NET

```
def newsProviders(self, newsProviders: ListOfNewsPr
        print("NewsProviders: ", newsProviders)
```

Live News Headlines

Important: in order to obtain news feeds via the TWS API you need to acquire the relevant API-specific subscriptions via your Account Management.

News articles provided through the API may not correspond to what is available directly through the Trader Workstation. Off-platform distribution of data is at the discretion of the news source provider, not by Interactive Brokers.

When invoking IBApi.EClient.reqMktData, for a specific IBApi.Contract you will follow the same format convention as any other basic contracts. The News Source is identified by the genericTickList argument.

Note: The error message "invalid tick type" will be returned if the username has not added the appropriate API news subscription.

Note: 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 API. For more information and to submit an API entitlement application, please contact Briefing.com directly at dbeasley@briefing.com.

Request Contract Specific News

EClient.reqMktData (

reqId: int. Request identifier for tracking data.

contract: Contract. Contract object used for specifying an instrument.

genericTickList: String. Comma separated ids of the available generic ticks.

snapshot: bool. Always set to false for news data.

regulatorySnapshot: bool. Always set to false for news data.

mktDataOptions: List<TagValue>. Internal use only.
)

Used to request market data typically, but can also be used to retrieve news. "mdoff" can be specified to disable standard market data while retrieving news. For news sources, genericTick 292 needs to be specified followed by a colon and the news provider's code.

Python | Java | C++ | C# | VB.NET

```
self.reqMktData(reqId, contract, "mdoff,292:BRFG",
```

Request BroadTape News

BroadTape News Contracts

For BroadTape News you specify the 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 EClientSocket.reqContractDetails request.

The symbol is typically the proivder code, a colon, then the news provider codes appended with "_ALL"

Python | Java | C++ | C# | VB.NET

Example news contract

```
contract = Contract()
contract.symbol  = "BRF:BRF_ALL"
contract.secType = "NEWS"
contract.exchange = "BRF"
```

EClient.reqMktData (

reqId: int. Request identifier for tracking data.

contract: Contract. Contract object used for specifying an instrument.

Python | Java | C++ | C# | VB.NET

```
self.reqMktData(reqId, contract, "mdoff,292", Falsa
```

genericTickList: String. Comma separated ids of the available generic ticks.

snapshot: bool. Always set to false for news data.

regulatorySnapshot: bool. Always set to false for news data.

mktDataOptions: List<TagValue>. Internal use only.
)

Used to request market data typically, but can also be used to retrieve news. "mdoff" can be specified to disable standard market data while retrieving news.

For news sources, genericTick 292 needs to be specified.

Receive Live News Headlines

EWrapper.tickNews (

tickerId: int. Request identifier used to track data.

timeStamp: int. Epoch time of the article's published time.

providerCode: String. News provider code based on requested data.

articleId: String. Identifier used to track the particular article. See News Article for more.

headline: String. Headline of the provided news article.

extraData: String. Returns any additional data available about the article.
)

Returns news headlines for requested contracts.

Python Java C++ C# VB.NET

```
def tickNews(self, tickerId: int, timeStamp: int,
  print("TickNews. TickerId:", tickerId, "TimeStam
```

Historical News Headlines

With the appropriate API news subscription, historical news headlines can be requested from the API using the function EClient::reqHistoricalNews. The resulting headlines are returned to EWrapper::historicalNews.

Requesting Historical News

EClient.reqHistoricalNews (

requestId: int. Request identifier used to track data.

conId: int. Contract id of ticker. See Contract Details for how to retrieve conId.

providerCodes: String. A '+'-separated list of provider codes.

startDateTime: String. Marks the (exclusive) start of the date range. The format is yyyy-MM-dd HH:mm:ss.

endDateTime: String. Marks the (inclusive) end of the date range. The format is yyyy-MM-dd HH:mm:ss.

totalResults: int. The maximum number of headlines to fetch (1 – 300)

historicalNewsOptions: Null. Reserved for internal use. Should be defined as null.
)

Requests historical news headlines.

Python Java C++ C# VB.NET

```
self.reqHistoricalNews(reqId, 8314, "BRFG", "", ""
```

Receive Historical News

EWrapper.historicalNews (

requestId: int. Request identifier used to track data.

time: int. Epoch time of the article's published time.

providerCode: String. News provider code based on requested data.

articleId: String. Identifier used to track the particular article. See News Article for more.

headline: String. Headline of the provided news article.
)

Returns news headlines for requested contracts.

Python Java C++ C# VB.NET

```
def historicalNews(self, requestId: int, time: int,
  print("historicalNews. RequestId:", requestId, "
```

EWrapper.historicalNewsEnd (

requestId: int. Request identifier used to track data.

hasMore: bool. Returns whether there is more data (true) or not (false).
)

Returns news headlines end marker

Python Java C++ C# VB.NET

```
def historicalDataEnd(self, reqId: int, hasMore: bo
    print("historicalDataEnd. ReqId:", reqId, "Has
```

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. The body of the news article is returned to the function IBApi::EWrapper::newsArticle.

Request News Articles

EClient.reqNewsArticle (

requestId: int. Id of the request.

providerCode: String. Short code indicating news provider, e.g. FLY.

articleId: String. Id of the specific article.

newsArticleOptions: List. Reserved for internal use. Should be defined as null.
)

Requests news article body given articleId.

Python Java C++ C# VB.NET

```
self.reqNewsArticle(10002,"BRFG", "BRFG$04fb9da2",
```

Receive News Articles

EWrapper.newsArticle (

requestId: int. Request identifier used to track data.

articleType: int. The type of news article (0 – plain text or html, 1 – binary data / pdf).

articleText: String. The body of article (if articleType == 1, the binary data is encoded using the Base64 scheme).
)

Called when receiving a News Article in response to reqNewsArticle().

Python Java C++ C# VB.NET

```
def newsArticle(self, requestId: int, articleType:
  print("requestId: ", requestId, "articleType: ",
```

Next Valid ID

Perhaps the most important event received after successfully connecting to the TWS is the EWrapper.nextValidId, which is also triggered after invoking the 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 EClient.reqIds method in the event that your client application loses track of the sequence.

Reset Order ID Sequence

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.

Request Next Valid ID

EClient.reqIds (

numIds: int. (Deprecated) This parameter will not affect the value returned to nextValidId but is required.
)

Requests the next valid order ID at the current moment.

Python | Java | C++ | C# | VB.NET

```
self.reqIds(-1)
```

Receive Next Valid ID

EWrapper.nextValidId (

orderId: int. 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.

Python | Java | C++ | C# | VB.NET

```
def nextValidId(self, orderId: int):
    print("NextValidId:", orderId)
```

TWS UI Display Groups

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.

Query Display Groups

The IBApi.EClient.queryDisplayGroups method is used to request all available Display Groups in TWS. The IBApi.EWrapper.displayGroupList is a one-time response to IBApi.EClient.queryDisplayGroups.

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"

Request Query Display Groups

EClient.queryDisplayGroups (

requestId: int. Request identifier used to track data.
)

Requests all available Display Groups in TWS.

Python | Java | C++ | C# | VB.NET

```
self.queryDisplayGroups(requestId)
```

Receive Query Display Groups

EWrapper.displayGroupList (

requestId: Request identifier used to track data.

groups: String. Returns a list of integers representing visible Group ID separated by the "|" character, and sorted by most used group first.
)

A one-time response to querying the display groups.

Python | Java | C++ | C# | VB.NET

```
def displayGroupList(self, reqId: int, groups: str
  print("DisplayGroupList. ReqId:", reqId, "Groups
```

Subscribe To Group Events

To integrate with a specific Group, you need to first subscribe to the group number by invoking IBApi.EClient.subscribeToGroupEvents. 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.

Request Group Events Subscription

EClient.subscribeToGroupEvents (

requestId: int. Request identifier used to track data.

groupId: int. The display group for integration.
)

Integrates API client and TWS window grouping.

Python | Java | C++ | C# | VB.NET

```
self.subscribeToGroupEvents(19002, 1)
```

Receive Group Events Subscription

EWrapper.displayGroupUpdated (

requestId: int. Request identifier used to track data.

contractInfo: String. Contract information produced for the active display group.

)
Call triggered once after receiving the subscription request, and will be sent again if the selected contract in the subscribed * display group has changed.

Python | Java | C++ | C# | VB.NET

```
def displayGroupUpdated(self, reqId: int, contract:
      print("DisplayGroupUpdated. ReqId:", reqId
```

Unsubscribe From Group Events

EClient.unsubscribeFromGroupEvents (

requestId: int. Request identifier used to track data.
)

Cancels a TWS Window Group subscription.

Python | Java | C++ | C# | VB.NET

```
self.unsubscribeFromGroupEvents(19002)
```

Update Display Group

EClient.updateDisplayGroup (

requestId: int. Request identifier used for tracking data.

contractInfo: String. An encoded value designating a unique IB contract. Possible values include:

- none: Empty selection
- contractID: Any non-combination contract. Examples 8314 for IBM SMART; 8314 for IBM ARCA
- combo: If any combo is selected Note: This request from the API does not get a TWS response unless an error occurs.

)

Updates the contract displayed in a TWS Window Group.
Note: This request from the API does not get a response from TWS unless an error occurs.

In this sample we have commanded TWS Windows that chained with Group #1 to display IBM@SMART. The screenshot of TWS Mosaic to the right shows that both the pink chained (Group #1) windows are now displaying IBM@SMART, while the green chained (Group #4) window remains unchanged.

Python Java C++ C# VB.NET

```
self.updateDisplayGroup(19002, "8314@SMART")
```

Wall Street Horizon

Calendar and Event data can be retrieved from the Wall Street Horizon Event Calendar and accessed via the TWS API through the functions IBApi.EClient.reqWshMetaData and IBApi.EClient.reqWshEventData.

It is necessary to have the **Wall Street Horizon Corporate Event Data** research subscription activated first in Account Management.
WSH provides IBKR with corporate event datasets, including earnings dates, dividend dates, options expiration dates, splits, spinoffs and a wide variety of investor-related conferences.

Data Classes and Fields PDF

Meta Data

The function IBApi.EClient.reqWshMetaData is used to request metadata describing calendar events.

Requesting Meta Data

EClient.reqWshMetaData (

requestId: int. Request identifier used to track data.
)

Requests metadata from the WSH calendar.

Python Java C++ C# VB.NET

```
self.reqWshMetaData(1100)
```

Receive Meta Data

EWrapper.wshEventData (

requestId: int. Request identifier used to track data.

dataJson: String. metadata in json format.
)

Returns meta data from the WSH calendar

Python Java C++ C# VB.NET

```
def wshMetaData(self, reqId: int, dataJson: str):
        print("WshMetaData.", "ReqId:", reqId, "Da
```

Cancel Meta Data

EClient.cancelWshMetaData (

requestId: int. Request identifier used to track data.
)

Cancels pending request for WSH metadata.

Python Java C++ C# VB.NET

```
self.cancelWshMetaData(1100)
```

Event Data

The function EClient.reqWshEventData is used to request the calendar events. The event data is then received via the callback EWrapper.wshEventData. Pending event data requests can be canceled with the function IBApi.EClient.cancelWshEventData.

Note: Prior to sending this message, it is expected that the API client request metadata via EClient.reqWshMetaData, else an error may be reported.

Also note that TWS will not support multiple concurrent requests. Previous request should succeed, fail, or be cancelled by client before next one. TWS will reject such requests with text "Duplicate WSH meta-data request" or "Duplicate WSH event request".

WshEventData Object

When making a request to the Wall Street Horizons Event Calendar with the API, users must create a wshEventData Object. This object contains several fields, along with a filter field, which takes a json-formatted string. The filter values are returned from WSH Meta Data requests.

When creating the object, users are able to specify either the WshEventData.conId, WshEventData.startDate, and WshEventData.endDate, or they may choose to use the WshEventData.filter value. Attempting to use both will result in an error.

The value to the right provides an outline of available fields, though new fields may be available.

Because Interactive Brokers are adding additional fields frequently, we always suggest reviewing the code in our latest releases, available in your TWS API installation.

WshEventData()

conId: String. Specify the contract identifier for the event request.

startDate: String. Specify the start date of the event requests. Formatted as "YYYYMMDD"

endDate: String. Specify the end date of the event requests. Formatted as "YYYYMMDD"

fillCompetitors: bool. Automatically fill in competitor values of existing positions.

fillPortfolio: bool. Automatically fill in portfolio values.

fillWatchlist: bool. Automatically fill in watchlist values.

totalLimit: int. Maximum of 100.

filter: String. Json-formatted string containing all filter values. Some available values include:

- watchlist: Array of string. Takes a single conid.
- country: String. Specify a country code, or "All".
- The header section of wshMetaData responses will include a value formatted as "wshe_". These values can be included or ignored in an eventData request with a true/false value.

Python Java C++ C# VB.NET

{TWS API}\source\pythonclient\ibapi\client.py

Request Event Data

EClient.reqWshEventData (

requestId: int. Request identifier used to track data.

Python Java C++ C# VB.NET

wshEventData: WshEventData. Unique object used to track all parameters for the event data request. See WshEventData Object for more details.
)

Requests event data from the WSH calendar.

```
self.reqWshEventData(1101, eventDataObj)
```

Receive Event Data

EWrapper.wshEventData(

requestId: int. Request identifier used to track data.

dataJson: String. Event data json format.
)

Returns calendar events from the WSH.

Python | Java | C++ | C# | VB.NET

```
def wshEventData(self, reqId: int, dataJson: str):
        print("WshEventData.", "ReqId:", reqId, "Da
```

Cancel Event Data

EClient.cancelWshEventData(

requestId: int. Request identifier used to track data.

)

Cancels pending WSH event data request.

Python | Java | C++ | C# | VB.NET

```
self.cancelWshEventData(1101, eventDataObj)
```