

---

**ib\_insync**

***Release 0.9.86***

**Ewald de Wit**

**Dec 11, 2023**



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Example . . . . .	3
1.3	Documentation . . . . .	4
1.4	Discussion . . . . .	4
1.5	Disclaimer . . . . .	4
<b>2</b>	<b>API docs</b>	<b>5</b>
2.1	IB . . . . .	5
2.2	Client . . . . .	30
2.3	Order . . . . .	35
2.4	Contract . . . . .	52
2.5	Ticker . . . . .	68
2.6	Objects . . . . .	75
2.7	Utilities . . . . .	94
2.8	FlexReport . . . . .	98
2.9	IBC . . . . .	98
2.10	Watchdog . . . . .	101
<b>3</b>	<b>Notebooks</b>	<b>105</b>
<b>4</b>	<b>Code recipes</b>	<b>107</b>
4.1	Fetching consecutive historical data . . . . .	107
4.2	Scanner data (blocking) . . . . .	108
4.3	Scanner data (streaming) . . . . .	108
4.4	Option calculations . . . . .	108
4.5	Order book . . . . .	109
4.6	Minimum price increments . . . . .	109
4.7	News articles . . . . .	109
4.8	News bulletins . . . . .	109
4.9	WSH Event Calendar . . . . .	110
4.10	Dividends . . . . .	110
4.11	Fundamental ratios . . . . .	111
4.12	Short-lived connections . . . . .	111
4.13	Integration with PyQt5 or PySide2 . . . . .	112
4.14	Integration with Tkinter . . . . .	112
4.15	Integration with PyGame . . . . .	113
<b>5</b>	<b>Source code</b>	<b>115</b>
<b>6</b>	<b>Changelog</b>	<b>117</b>

6.1	0.9	117
6.2	0.8	130
6.3	0.7	133
6.4	0.6	134
<b>7</b>	<b>Links</b>	<b>135</b>
<b>8</b>	<b>Introduction</b>	<b>137</b>
8.1	Installation	137
8.2	Example	137
8.3	Documentation	138
8.4	Discussion	138
8.5	Disclaimer	138
	<b>Python Module Index</b>	<b>139</b>
	<b>Index</b>	<b>141</b>





## INTRODUCTION

The goal of the IB-insync library is to make working with the [Trader Workstation API](#) from Interactive Brokers as easy as possible.

The main features are:

- An easy to use linear style of programming;
- An [IB component](#) that automatically keeps in sync with the TWS or IB Gateway application;
- A fully asynchronous framework based on [asyncio](#) and [eventkit](#) for advanced users;
- Interactive operation with live data in Jupyter notebooks.

Be sure to take a look at the [notebooks](#), the [recipes](#) and the [API docs](#).

### 1.1 Installation

```
pip install ib_insync
```

Requirements:

- Python 3.6 or higher;
- A running TWS or IB Gateway application (version 1023 or higher). Make sure the [API port is enabled](#) and 'Download open orders on connection' is checked.

The ibapi package from IB is not needed.

### 1.2 Example

This is a complete script to download historical data:

```
from ib_insync import *
# util.startLoop() # uncomment this line when in a notebook

ib = IB()
ib.connect('127.0.0.1', 7497, clientId=1)

contract = Forex('EURUSD')
bars = ib.reqHistoricalData(
    contract, endDateTime='', durationStr='30 D',
```

(continues on next page)

(continued from previous page)

```
barSizeSetting='1 hour', whatToShow='MIDPOINT', useRTH=True)

# convert to pandas dataframe (pandas needs to be installed):
df = util.df(bars)
print(df)
```

Output:

	date	open	high	low	close	volume	\
0	2019-11-19 23:15:00	1.107875	1.108050	1.107725	1.107825	-1	
1	2019-11-20 00:00:00	1.107825	1.107925	1.107675	1.107825	-1	
2	2019-11-20 01:00:00	1.107825	1.107975	1.107675	1.107875	-1	
3	2019-11-20 02:00:00	1.107875	1.107975	1.107025	1.107225	-1	
4	2019-11-20 03:00:00	1.107225	1.107725	1.107025	1.107525	-1	
...	...	...	...	...	...	...	
705	2020-01-02 14:00:00	1.119325	1.119675	1.119075	1.119225	-1	

## 1.3 Documentation

The complete API documentation.

Changelog.

## 1.4 Discussion

The [insync user group](#) is the place to discuss IB-insync and anything related to it.

## 1.5 Disclaimer

The software is provided on the conditions of the simplified BSD license.

This project is not affiliated with Interactive Brokers Group, Inc.'s.

Good luck and enjoy.

**author**

Ewald de Wit <[ewald.de.wit@gmail.com](mailto:ewald.de.wit@gmail.com)>



Release 0.9.86.

Also see the official [Python API documentation](#) from IB.

## 2.1 IB

High-level interface to Interactive Brokers.

**class** `ib_insync.ib.IB`

Provides both a blocking and an asynchronous interface to the IB API, using asyncio networking and event loop.

The IB class offers direct access to the current state, such as orders, executions, positions, tickers etc. This state is automatically kept in sync with the TWS/IBG application.

This class has most request methods of EClient, with the same names and parameters (except for the reqId parameter which is not needed anymore). Request methods that return a result come in two versions:

- Blocking: Will block until complete and return the result. The current state will be kept updated while the request is ongoing;
- Asynchronous: All methods that have the “Async” postfix. Implemented as coroutines or methods that return a Future and intended for advanced users.

### **The One Rule:**

While some of the request methods are blocking from the perspective of the user, the framework will still keep spinning in the background and handle all messages received from TWS/IBG. It is important to not block the framework from doing its work. If, for example, the user code spends much time in a calculation, or uses `time.sleep()` with a long delay, the framework will stop spinning, messages accumulate and things may go awry.

The one rule when working with the IB class is therefore that

**user code may not block for too long.**

To be clear, the IB request methods are okay to use and do not count towards the user operation time, no matter how long the request takes to finish.

So what is “too long”? That depends on the situation. If, for example, the timestamp of tick data is to remain accurate within a millisecond, then the user code must not spend longer than a millisecond. If, on the other extreme, there is very little incoming data and there is no desire for accurate timestamps, then the user code can block for hours.

If a user operation takes a long time then it can be farmed out to a different process. Alternatively the operation can be made such that it periodically calls `IB.sleep(0)`; This will let the framework handle any pending work and return when finished. The operation should be aware that the current state may have been updated during the `sleep(0)` call.

For introducing a delay, never use `time.sleep()` but use `sleep()` instead.

### Parameters

- **RequestTimeout** (*float*) – Timeout (in seconds) to wait for a blocking request to finish before raising `asyncio.TimeoutError`. The default value of 0 will wait indefinitely. Note: This timeout is not used for the `*Async` methods.
- **RaiseRequestErrors** (*bool*) – Specifies the behaviour when certain API requests fail:
  - **False**: Silently return an empty result;
  - **True**: Raise a `RequestError`.
- **MaxSyncedSubAccounts** (*int*) – Do not use sub-account updates if the number of sub-accounts exceeds this number (50 by default).
- **TimezoneTWS** (*str*) – Specifies what timezone TWS (or gateway) is using. The default is to assume local system timezone.

### Events:

- `connectedEvent ()`: Is emitted after connecting and synchronizing with TWS/gateway.
- `disconnectedEvent ()`: Is emitted after disconnecting from TWS/gateway.
- `updateEvent ()`: Is emitted after a network packet has been handled.
- `pendingTickersEvent (tickers: Set[Ticker])`: Emits the set of tickers that have been updated during the last update and for which there are new ticks, `tickByTicks` or `domTicks`.
- `barUpdateEvent (bars: BarDataList, hasNewBar: bool)`: Emits the bar list that has been updated in real time. If a new bar has been added then `hasNewBar` is `True`, when the last bar has changed it is `False`.
- `newOrderEvent (trade: Trade)`: Emits a newly placed trade.
- `orderModifyEvent (trade: Trade)`: Emits when order is modified.
- `cancelOrderEvent (trade: Trade)`: Emits a trade directly after requesting for it to be cancelled.
- `openOrderEvent (trade: Trade)`: Emits the trade with open order.
- `orderStatusEvent (trade: Trade)`: Emits the changed order status of the ongoing trade.
- `execDetailsEvent (trade: Trade, fill: Fill)`: Emits the fill together with the ongoing trade it belongs to.
- `commissionReportEvent (trade: Trade, fill: Fill, report: CommissionReport)`: The commission report is emitted after the fill that it belongs to.
- `updatePortfolioEvent (item: PortfolioItem)`: A portfolio item has changed.
- `positionEvent (position: Position)`: A position has changed.
- `accountValueEvent (value: AccountValue)`: An account value has changed.
- `accountSummaryEvent (value: AccountValue)`: An account value has changed.
- `pnlEvent (entry: PnL)`: A profit- and loss entry is updated.
- `pnlSingleEvent (entry: PnLSingle)`: A profit- and loss entry for a single position is updated.
- `tickNewsEvent (news: NewsTick)`: Emit a new news headline.
- `newsBulletinEvent (bulletin: NewsBulletin)`: Emit a new news bulletin.
- `scannerDataEvent (data: ScanDataList)`: Emit data from a scanner subscription.

- `wshMetaEvent` (dataJson: str): Emit WSH metadata.
- `wshEvent` (dataJson: str): Emit WSH event data (such as earnings dates, dividend dates, options expiration dates, splits, spinoffs and conferences).
- `errorEvent` (reqId: int, errorCode: int, errorString: str, contract: *Contract*): Emits the reqId/orderId and TWS error code and string (see [https://interactivebrokers.github.io/tws-api/message\\_codes.html](https://interactivebrokers.github.io/tws-api/message_codes.html)) together with the contract the error applies to (or None if no contract applies).
- `timeoutEvent` (idlePeriod: float): Is emitted if no data is received for longer than the timeout period specified with `setTimeout()`. The value emitted is the period in seconds since the last update.

Note that it is not advisable to place new requests inside an event handler as it may lead to too much recursion.

```
events = ('connectedEvent', 'disconnectedEvent', 'updateEvent',
'pendingTickersEvent', 'barUpdateEvent', 'newOrderEvent', 'orderModifyEvent',
'cancelOrderEvent', 'openOrderEvent', 'orderStatusEvent', 'execDetailsEvent',
'commissionReportEvent', 'updatePortfolioEvent', 'positionEvent',
'accountValueEvent', 'accountSummaryEvent', 'pnlEvent', 'pnlSingleEvent',
'scannerDataEvent', 'tickNewsEvent', 'newsBulletinEvent', 'wshMetaEvent',
'wshEvent', 'errorEvent', 'timeoutEvent')
```

```
RequestTimeout: float = 0
```

```
RaiseRequestErrors: bool = False
```

```
MaxSyncedSubAccounts: int = 50
```

```
TimezoneTWS: str = ''
```

```
connect(host='127.0.0.1', port=7497, clientId=1, timeout=4, readonly=False, account='',
        raiseSyncErrors=False)
```

Connect to a running TWS or IB gateway application. After the connection is made the client is fully synchronized and ready to serve requests.

This method is blocking.

#### Parameters

- **host** (str) – Host name or IP address.
- **port** (int) – Port number.
- **clientId** (int) – ID number to use for this client; must be unique per connection. Setting clientId=0 will automatically merge manual TWS trading with this client.
- **timeout** (float) – If establishing the connection takes longer than timeout seconds then the `asyncio.TimeoutError` exception is raised. Set to 0 to disable timeout.
- **readonly** (bool) – Set to True when API is in read-only mode.
- **account** (str) – Main account to receive updates for.
- **raiseSyncErrors** (bool) – When True this will cause an initial sync request error to raise a `ConnectionError`. When False the error will only be logged at error level.

#### disconnect()

Disconnect from a TWS or IB gateway application. This will clear all session state.

**isConnected()**

Is there an API connection to TWS or IB gateway?

**Return type**

`bool`

**static run(\*, timeout=None)**

By default run the event loop forever.

When awaitables (like Tasks, Futures or coroutines) are given then run the event loop until each has completed and return their results.

An optional timeout (in seconds) can be given that will raise `asyncio.TimeoutError` if the awaitables are not ready within the timeout period.

**static schedule(callback, \*args)**

Schedule the callback to be run at the given time with the given arguments. This will return the Event Handle.

**Parameters**

- **time** (`Union[time, datetime]`) – Time to run callback. If given as `datetime.time` then use today as date.
- **callback** (`Callable`) – Callable scheduled to run.
- **args** – Arguments for to call callback with.

**static sleep()**

Wait for the given amount of seconds while everything still keeps processing in the background. Never use `time.sleep()`.

**Parameters**

**secs** (`float`) – Time in seconds to wait.

**Return type**

`bool`

**static timeRange(end, step)**

Iterator that waits periodically until certain time points are reached while yielding those time points.

**Parameters**

- **start** (`Union[time, datetime]`) – Start time, can be specified as `datetime.datetime`, or as `datetime.time` in which case today is used as the date
- **end** (`Union[time, datetime]`) – End time, can be specified as `datetime.datetime`, or as `datetime.time` in which case today is used as the date
- **step** (`float`) – The number of seconds of each period

**Return type**

`Iterator[datetime]`

**static timeRangeAsync(end, step)**

Async version of `timeRange()`.

**Return type**

`AsyncIterator[datetime]`

**static waitUntil()**

Wait until the given time `t` is reached.

**Parameters**

**t** (`Union[time, datetime]`) – The time *t* can be specified as `datetime.datetime`, or as `datetime.time` in which case today is used as the date.

**Return type**

`bool`

**waitOnUpdate**(*timeout=0*)

Wait on any new update to arrive from the network.

**Parameters**

**timeout** (`float`) – Maximum time in seconds to wait. If 0 then no timeout is used.

---

**Note:** A loop with `waitOnUpdate` should not be used to harvest tick data from tickers, since some ticks can go missing. This happens when multiple updates occur almost simultaneously; The ticks from the first update are then cleared. Use events instead to prevent this.

---

**Return type**

`bool`

**Returns**

True if not timed-out, False otherwise.

**loopUntil**(*condition=None, timeout=0*)

Iterate until condition is met, with optional timeout in seconds. The yielded value is that of the condition or False when timed out.

**Parameters**

- **condition** – Predicate function that is tested after every network
- **update.** –
- **timeout** (`float`) – Maximum time in seconds to wait. If 0 then no timeout is used.

**Return type**

`Iterator[object]`

**setTimeout**(*timeout=60*)

Set a timeout for receiving messages from TWS/IBG, emitting `timeoutEvent` if there is no incoming data for too long.

The timeout fires once per connected session but can be set again after firing or after a reconnect.

**Parameters**

**timeout** (`float`) – Timeout in seconds.

**managedAccounts**()

List of account names.

**Return type**

`List[str]`

**accountValues**(*account=""*)

List of account values for the given account, or of all accounts if account is left blank.

**Parameters**

**account** (`str`) – If specified, filter for this account name.

**Return type**`List[AccountValue]`**accountSummary**(*account=""*)

List of account values for the given account, or of all accounts if account is left blank.

This method is blocking on first run, non-blocking after that.

**Parameters**

**account** (*str*) – If specified, filter for this account name.

**Return type**`List[AccountValue]`**portfolio**(*account=""*)

List of portfolio items for the given account, or of all retrieved portfolio items if account is left blank.

**Parameters**

**account** (*str*) – If specified, filter for this account name.

**Return type**`List[PortfolioItem]`**positions**(*account=""*)

List of positions for the given account, or of all accounts if account is left blank.

**Parameters**

**account** (*str*) – If specified, filter for this account name.

**Return type**`List[Position]`**pnl**(*account="", modelCode=""*)

List of subscribed *PnL* objects (profit and loss), optionally filtered by account and/or modelCode.

The *PnL* objects are kept live updated.

**Parameters**

- **account** – If specified, filter for this account name.
- **modelCode** – If specified, filter for this account model.

**Return type**`List[PnL]`**pnlSingle**(*account="", modelCode="", conId=0*)

List of subscribed *PnLSingle* objects (profit and loss for single positions).

The *PnLSingle* objects are kept live updated.

**Parameters**

- **account** (*str*) – If specified, filter for this account name.
- **modelCode** (*str*) – If specified, filter for this account model.
- **conId** (*int*) – If specified, filter for this contract ID.

**Return type**`List[PnLSingle]`

**trades()**

List of all order trades from this session.

**Return type**

`List[Trade]`

**openTrades()**

List of all open order trades.

**Return type**

`List[Trade]`

**orders()**

List of all orders from this session.

**Return type**

`List[Order]`

**openOrders()**

List of all open orders.

**Return type**

`List[Order]`

**fills()**

List of all fills from this session.

**Return type**

`List[Fill]`

**executions()**

List of all executions from this session.

**Return type**

`List[Execution]`

**ticker(contract)**

Get ticker of the given contract. It must have been requested before with `reqMktData` with the same contract object. The ticker may not be ready yet if called directly after `reqMktData()`.

**Parameters**

**contract** (`Contract`) – Contract to get ticker for.

**Return type**

`Optional[Ticker]`

**tickers()**

Get a list of all tickers.

**Return type**

`List[Ticker]`

**pendingTickers()**

Get a list of all tickers that have pending ticks or domTicks.

**Return type**

`List[Ticker]`

**realtimeBars()**

Get a list of all live updated bars. These can be 5 second realtime bars or live updated historical bars.

**Return type**`List[Union[BarDataList, RealTimeBarList]]`**newsTicks()**

List of ticks with headline news. The article itself can be retrieved with `reqNewsArticle()`.

**Return type**`List[NewsTick]`**newsBulletins()**

List of IB news bulletins.

**Return type**`List[NewsBulletin]`**reqTickers(\*contracts, regulatorySnapshot=False)**

Request and return a list of snapshot tickers. The list is returned when all tickers are ready.

This method is blocking.

**Parameters**

- **contracts** (`Contract`) – Contracts to get tickers for.
- **regulatorySnapshot** (`bool`) – Request NBBO snapshots (may incur a fee).

**Return type**`List[Ticker]`**qualifyContracts(\*contracts)**

Fully qualify the given contracts in-place. This will fill in the missing fields in the contract, especially the `conId`.

Returns a list of contracts that have been successfully qualified.

This method is blocking.

**Parameters**

**contracts** (`Contract`) – Contracts to qualify.

**Return type**`List[Contract]`**bracketOrder(action, quantity, limitPrice, takeProfitPrice, stopLossPrice, \*\*kwargs)**

Create a limit order that is bracketed by a take-profit order and a stop-loss order. Submit the bracket like:

```
for o in bracket:
    ib.placeOrder(contract, o)
```

[https://interactivebrokers.github.io/tws-api/bracket\\_order.html](https://interactivebrokers.github.io/tws-api/bracket_order.html)

**Parameters**

- **action** (`str`) – ‘BUY’ or ‘SELL’.
- **quantity** (`float`) – Size of order.
- **limitPrice** (`float`) – Limit price of entry order.
- **takeProfitPrice** (`float`) – Limit price of profit order.
- **stopLossPrice** (`float`) – Stop price of loss order.

**Return type**`BracketOrder`



**static oneCancelsAll**(*orders, ocaGroup, ocaType*)

Place the trades in the same One Cancels All (OCA) group.

<https://interactivebrokers.github.io/tws-api/oca.html>

**Parameters**

**orders** (*List[Order]*) – The orders that are to be placed together.

**Return type**

*List[Order]*

**whatIfOrder**(*contract, order*)

Retrieve commission and margin impact without actually placing the order. The given order will not be modified in any way.

This method is blocking.

**Parameters**

- **contract** (*Contract*) – Contract to test.
- **order** (*Order*) – Order to test.

**Return type**

*OrderState*

**placeOrder**(*contract, order*)

Place a new order or modify an existing order. Returns a Trade that is kept live updated with status changes, fills, etc.

**Parameters**

- **contract** (*Contract*) – Contract to use for order.
- **order** (*Order*) – The order to be placed.

**Return type**

*Trade*

**cancelOrder**(*order, manualCancelOrderTime=""*)

Cancel the order and return the Trade it belongs to.

**Parameters**

- **order** (*Order*) – The order to be canceled.
- **manualCancelOrderTime** (*str*) – For audit trail.

**Return type**

*Optional[Trade]*

**reqGlobalCancel**()

Cancel all active trades including those placed by other clients or TWS/IB gateway.

**reqCurrentTime**()

Request TWS current time.

This method is blocking.

**Return type**

*datetime*

**reqAccountUpdates**(*account=""*)

This is called at startup - no need to call again.

Request account and portfolio values of the account and keep updated. Returns when both account values and portfolio are filled.

This method is blocking.

**Parameters**

**account** (*str*) – If specified, filter for this account name.

**reqAccountUpdatesMulti**(*account="", modelCode=""*)

It is recommended to use [\*accountValues\(\)\*](#) instead.

Request account values of multiple accounts and keep updated.

This method is blocking.

**Parameters**

- **account** (*str*) – If specified, filter for this account name.
- **modelCode** (*str*) – If specified, filter for this account model.

**reqAccountSummary**()

It is recommended to use [\*accountSummary\(\)\*](#) instead.

Request account values for all accounts and keep them updated. Returns when account summary is filled.

This method is blocking.

**reqAutoOpenOrders**(*autoBind=True*)

Bind manual TWS orders so that they can be managed from this client. The *clientId* must be 0 and the TWS API setting “Use negative numbers to bind automatic orders” must be checked.

This request is automatically called when *clientId*=0.

[https://interactivebrokers.github.io/tws-api/open\\_orders.html](https://interactivebrokers.github.io/tws-api/open_orders.html) [https://interactivebrokers.github.io/tws-api/modifying\\_orders.html](https://interactivebrokers.github.io/tws-api/modifying_orders.html)

**Parameters**

**autoBind** (*bool*) – Set binding on or off.

**reqOpenOrders**()

Request and return a list of open orders.

This method can give stale information where a new open order is not reported or an already filled or cancelled order is reported as open. It is recommended to use the more reliable and much faster [\*openTrades\(\)\*](#) or [\*openOrders\(\)\*](#) methods instead.

This method is blocking.

**Return type**

[\*List\[Trade\]\*](#)

**reqAllOpenOrders**()

Request and return a list of all open orders over all clients. Note that the orders of other clients will not be kept in sync, use the master *clientId* mechanism instead to see other client’s orders that are kept in sync.

**Return type**

[\*List\[Trade\]\*](#)

**reqCompletedOrders(*apiOnly*)**

Request and return a list of completed trades.

**Parameters**

**apiOnly** (*bool*) – Request only API orders (not manually placed TWS orders).

**Return type**

*List[Trade]*

**reqExecutions(*execFilter=None*)**

It is recommended to use *fills()* or *executions()* instead.

Request and return a list of fills.

This method is blocking.

**Parameters**

**execFilter** (*Optional[ExecutionFilter]*) – If specified, return executions that match the filter.

**Return type**

*List[Fill]*

**reqPositions()**

It is recommended to use *positions()* instead.

Request and return a list of positions for all accounts.

This method is blocking.

**Return type**

*List[Position]*

**reqPnL(*account*, *modelCode=""*)**

Start a subscription for profit and loss events.

Returns a *PnL* object that is kept live updated. The result can also be queried from *pnl()*.

<https://interactivebrokers.github.io/tws-api/pnl.html>

**Parameters**

- **account** (*str*) – Subscribe to this account.
- **modelCode** (*str*) – If specified, filter for this account model.

**Return type**

*PnL*

**cancelPnL(*account*, *modelCode=""*)**

Cancel PnL subscription.

**Parameters**

- **account** – Cancel for this account.
- **modelCode** (*str*) – If specified, cancel for this account model.

**reqPnLSingle(*account*, *modelCode*, *conId*)**

Start a subscription for profit and loss events for single positions.

Returns a *PnLSingle* object that is kept live updated. The result can also be queried from *pnlSingle()*.

<https://interactivebrokers.github.io/tws-api/pnl.html>

**Parameters**

- **account** (*str*) – Subscribe to this account.
- **modelCode** (*str*) – Filter for this account model.
- **conId** (*int*) – Filter for this contract ID.

**Return type***PnLSingle***cancelPnLSingle**(*account*, *modelCode*, *conId*)

Cancel PnLSingle subscription for the given account, modelCode and conId.

**Parameters**

- **account** (*str*) – Cancel for this account name.
- **modelCode** (*str*) – Cancel for this account model.
- **conId** (*int*) – Cancel for this contract ID.

**reqContractDetails**(*contract*)

Get a list of contract details that match the given contract. If the returned list is empty then the contract is not known; If the list has multiple values then the contract is ambiguous.

The fully qualified contract is available in the the `ContractDetails.contract` attribute.

This method is blocking.

[https://interactivebrokers.github.io/tws-api/contract\\_details.html](https://interactivebrokers.github.io/tws-api/contract_details.html)

**Parameters**

**contract** (*Contract*) – The contract to get details for.

**Return type***List[ContractDetails]***reqMatchingSymbols**(*pattern*)

Request contract descriptions of contracts that match a pattern.

This method is blocking.

[https://interactivebrokers.github.io/tws-api/matching\\_symbols.html](https://interactivebrokers.github.io/tws-api/matching_symbols.html)

**Parameters**

**pattern** (*str*) – The first few letters of the ticker symbol, or for longer strings a character sequence matching a word in the security name.

**Return type***List[ContractDescription]***reqMarketRule**(*marketRuleId*)

Request price increments rule.

[https://interactivebrokers.github.io/tws-api/minimum\\_increment.html](https://interactivebrokers.github.io/tws-api/minimum_increment.html)

**Parameters**

**marketRuleId** (*int*) – ID of market rule. The market rule IDs for a contract can be obtained via `reqContractDetails()` from `ContractDetails.marketRuleIds`, which contains a comma separated string of market rule IDs.

**Return type***PriceIncrement*

**reqRealTimeBars**(*contract*, *barSize*, *whatToShow*, *useRTH*, *realTimeBarsOptions*=[])

Request realtime 5 second bars.

[https://interactivebrokers.github.io/tws-api/realtime\\_bars.html](https://interactivebrokers.github.io/tws-api/realtime_bars.html)

#### Parameters

- **contract** (*Contract*) – Contract of interest.
- **barSize** (*int*) – Must be 5.
- **whatToShow** (*str*) – Specifies the source for constructing bars. Can be ‘TRADES’, ‘MID-POINT’, ‘BID’ or ‘ASK’.
- **useRTH** (*bool*) – If True then only show data from within Regular Trading Hours, if False then show all data.
- **realTimeBarsOptions** (*List[TagValue]*) – Unknown.

#### Return type

*RealTimeBarList*

**cancelRealTimeBars**(*bars*)

Cancel the realtime bars subscription.

#### Parameters

**bars** (*RealTimeBarList*) – The bar list that was obtained from reqRealTimeBars.

**reqHistoricalData**(*contract*, *endDateTime*, *durationStr*, *barSizeSetting*, *whatToShow*, *useRTH*, *formatDate=1*, *keepUpToDate=False*, *chartOptions=[]*, *timeout=60*)

Request historical bar data.

This method is blocking.

[https://interactivebrokers.github.io/tws-api/historical\\_bars.html](https://interactivebrokers.github.io/tws-api/historical_bars.html)

#### Parameters

- **contract** (*Contract*) – Contract of interest.
- **endDateTime** (*Union[datetime, date, str, None]*) – Can be set to “” to indicate the current time, or it can be given as a datetime.date or datetime.datetime, or it can be given as a string in ‘yyyyMMdd HH:mm:ss’ format. If no timezone is given then the TWS login timezone is used.
- **durationStr** (*str*) – Time span of all the bars. Examples: ‘60 S’, ‘30 D’, ‘13 W’, ‘6 M’, ‘10 Y’.
- **barSizeSetting** (*str*) – Time period of one bar. Must be one of: ‘1 secs’, ‘5 secs’, ‘10 secs’, ‘15 secs’, ‘30 secs’, ‘1 min’, ‘2 mins’, ‘3 mins’, ‘5 mins’, ‘10 mins’, ‘15 mins’, ‘20 mins’, ‘30 mins’, ‘1 hour’, ‘2 hours’, ‘3 hours’, ‘4 hours’, ‘8 hours’, ‘1 day’, ‘1 week’, ‘1 month’.
- **whatToShow** (*str*) – Specifies the source for constructing bars. Must be one of: ‘TRADES’, ‘MIDPOINT’, ‘BID’, ‘ASK’, ‘BID\_ASK’, ‘ADJUSTED\_LAST’, ‘HISTORICAL\_VOLATILITY’, ‘OPTION\_IMPLIED\_VOLATILITY’, ‘REBATE\_RATE’, ‘FEE\_RATE’, ‘YIELD\_BID’, ‘YIELD\_ASK’, ‘YIELD\_BID\_ASK’, ‘YIELD\_LAST’. For ‘SCHEDULE’ use *reqHistoricalSchedule()*.
- **useRTH** (*bool*) – If True then only show data from within Regular Trading Hours, if False then show all data.

- **formatDate** (*int*) – For an intraday request setting to 2 will cause the returned date fields to be timezone-aware `datetime.datetime` with UTC timezone, instead of local timezone as used by TWS.
- **keepUpToDate** (*bool*) – If True then a realtime subscription is started to keep the bars updated; `endTime` must be set empty (‘’) then.
- **chartOptions** (*List[TagValue]*) – Unknown.
- **timeout** (*float*) – Timeout in seconds after which to cancel the request and return an empty bar series. Set to 0 to wait indefinitely.

**Return type***BarDataList***cancelHistoricalData**(*bars*)

Cancel the update subscription for the historical bars.

**Parameters**

**bars** (*BarDataList*) – The bar list that was obtained from `reqHistoricalData` with a `keepUpToDate` subscription.

**reqHistoricalSchedule**(*contract, numDays, endTime='', useRTH=True*)

Request historical schedule.

This method is blocking.

**Parameters**

- **contract** (*Contract*) – Contract of interest.
- **numDays** (*int*) – Number of days.
- **endTime** (*Union[datetime, date, str, None]*) – Can be set to ‘’ to indicate the current time, or it can be given as a `datetime.date` or `datetime.datetime`, or it can be given as a string in ‘`yyyyMMdd HH:mm:ss`’ format. If no timezone is given then the TWS login timezone is used.
- **useRTH** (*bool*) – If True then show schedule for Regular Trading Hours, if False then for extended hours.

**Return type***HistoricalSchedule***reqHistoricalTicks**(*contract, startDateTime, endTime, numberOfTicks, whatToShow, useRth, ignoreSize=False, miscOptions=[]*)

Request historical ticks. The time resolution of the ticks is one second.

This method is blocking.

[https://interactivebrokers.github.io/tws-api/historical\\_time\\_and\\_sales.html](https://interactivebrokers.github.io/tws-api/historical_time_and_sales.html)

**Parameters**

- **contract** (*Contract*) – Contract to query.
- **startDateTime** (*Union[str, date]*) – Can be given as a `datetime.date` or `datetime.datetime`, or it can be given as a string in ‘`yyyyMMdd HH:mm:ss`’ format. If no timezone is given then the TWS login timezone is used.
- **endTime** (*Union[str, date]*) – One of `startDateTime` or `endTime` can be given, the other must be blank.

- **numberOfTicks** (*int*) – Number of ticks to request (1000 max). The actual result can contain a bit more to accommodate all ticks in the latest second.
- **whatToShow** (*str*) – One of 'Bid\_Ask', 'Midpoint' or 'Trades'.
- **useRTH** – If True then only show data from within Regular Trading Hours, if False then show all data.
- **ignoreSize** (*bool*) – Ignore bid/ask ticks that only update the size.
- **miscOptions** (*List[TagValue]*) – Unknown.

**Return type***List***reqMarketDataType**(*marketDataType*)

Set the market data type used for *reqMktData()*.

**Parameters**

**marketDataType** (*int*) – One of:

- 1 = Live
- 2 = Frozen
- 3 = Delayed
- 4 = Delayed frozen

[https://interactivebrokers.github.io/tws-api/market\\_data\\_type.html](https://interactivebrokers.github.io/tws-api/market_data_type.html)

**reqHeadTimeStamp**(*contract, whatToShow, useRTH, formatDate=1*)

Get the datetime of earliest available historical data for the contract.

**Parameters**

- **contract** (*Contract*) – Contract of interest.
- **useRTH** (*bool*) – If True then only show data from within Regular Trading Hours, if False then show all data.
- **formatDate** (*int*) – If set to 2 then the result is returned as a timezone-aware *datetime.datetime* with UTC timezone.

**Return type***datetime***reqMktData**(*contract, genericTickList="", snapshot=False, regulatorySnapshot=False, mktDataOptions=[]*)

Subscribe to tick data or request a snapshot. Returns the Ticker that holds the market data. The ticker will initially be empty and gradually (after a couple of seconds) be filled.

[https://interactivebrokers.github.io/tws-api/md\\_request.html](https://interactivebrokers.github.io/tws-api/md_request.html)

**Parameters**

- **contract** (*Contract*) – Contract of interest.
- **genericTickList** (*str*) – Comma separated IDs of desired generic ticks that will cause corresponding Ticker fields to be filled:

ID	Ticker fields
100	putVolume, callVolume (for options)
101	putOpenInterest, callOpenInterest (for options)
104	histVolatility (for options)
105	avOptionVolume (for options)
106	impliedVolatility (for options)
162	indexFuturePremium
165	low13week, high13week, low26week, high26week, low52week, high52week, avVolume
221	markPrice
225	auctionVolume, auctionPrice, auctionImbalance
233	last, lastSize, rtVolume, rtTime, vwap (Time & Sales)
236	shortableShares
258	fundamentalRatios (of type <i>ib_insync.objects.FundamentalRatios</i> )
293	tradeCount
294	tradeRate
295	volumeRate
375	rtTradeVolume
411	rtHistVolatility
456	dividends (of type <i>ib_insync.objects.Dividends</i> )
588	futuresOpenInterest

- **snapshot** (*bool*) – If True then request a one-time snapshot, otherwise subscribe to a stream of realtime tick data.
- **regulatorySnapshot** (*bool*) – Request NBBO snapshot (may incur a fee).
- **mktDataOptions** (*List[TagValue]*) – Unknown

**Return type***Ticker***cancelMktData**(*contract*)

Unsubscribe from realtime streaming tick data.

**Parameters****contract** (*Contract*) – The exact contract object that was used to subscribe with.**reqTickByTickData**(*contract, tickType, numberOfTicks=0, ignoreSize=False*)

Subscribe to tick-by-tick data and return the Ticker that holds the ticks in ticker.tickByTicks.

[https://interactivebrokers.github.io/tws-api/tick\\_data.html](https://interactivebrokers.github.io/tws-api/tick_data.html)**Parameters**

- **contract** (*Contract*) – Contract of interest.
- **tickType** (*str*) – One of ‘Last’, ‘AllLast’, ‘BidAsk’ or ‘MidPoint’.
- **numberOfTicks** (*int*) – Number of ticks or 0 for unlimited.
- **ignoreSize** (*bool*) – Ignore bid/ask ticks that only update the size.

**Return type***Ticker***cancelTickByTickData**(*contract, tickType*)

Unsubscribe from tick-by-tick data



**Parameters**

**contract** (*Contract*) – The exact contract object that was used to subscribe with.

**reqSmartComponents**(*bboExchange*)

Obtain mapping from single letter codes to exchange names.

Note: The exchanges must be open when using this request, otherwise an empty list is returned.

**Return type**

*List[SmartComponent]*

**reqMktDepthExchanges**()

Get those exchanges that have multiple market makers (and have ticks returned with marketMaker info).

**Return type**

*List[DepthMktDataDescription]*

**reqMktDepth**(*contract*, *numRows=5*, *isSmartDepth=False*, *mktDepthOptions=None*)

Subscribe to market depth data (a.k.a. DOM, L2 or order book).

[https://interactivebrokers.github.io/tws-api/market\\_depth.html](https://interactivebrokers.github.io/tws-api/market_depth.html)

**Parameters**

- **contract** (*Contract*) – Contract of interest.
- **numRows** (*int*) – Number of depth level on each side of the order book (5 max).
- **isSmartDepth** (*bool*) – Consolidate the order book across exchanges.
- **mktDepthOptions** – Unknown.

**Return type**

*Ticker*

**Returns**

The Ticker that holds the market depth in `ticker.domBids` and `ticker.domAsks` and the list of MktDepthData in `ticker.domTicks`.

**cancelMktDepth**(*contract*, *isSmartDepth=False*)

Unsubscribe from market depth data.

**Parameters**

**contract** (*Contract*) – The exact contract object that was used to subscribe with.

**reqHistogramData**(*contract*, *useRTH*, *period*)

Request histogram data.

This method is blocking.

<https://interactivebrokers.github.io/tws-api/histograms.html>

**Parameters**

- **contract** (*Contract*) – Contract to query.
- **useRTH** (*bool*) – If True then only show data from within Regular Trading Hours, if False then show all data.
- **period** (*str*) – Period of which data is being requested, for example '3 days'.

**Return type**

*List[HistogramData]*

**reqFundamentalData**(*contract*, *reportType*, *fundamentalDataOptions*=[])

Get fundamental data of a contract in XML format.

This method is blocking.

<https://interactivebrokers.github.io/tws-api/fundamentals.html>

**Parameters**

- **contract** (*Contract*) – Contract to query.
- **reportType** (*str*) –
  - ‘ReportsFinSummary’: Financial summary
  - ‘ReportsOwnership’: Company’s ownership
  - ‘ReportSnapshot’: Company’s financial overview
  - ‘ReportsFinStatements’: Financial Statements
  - ‘RESC’: Analyst Estimates
  - ‘CalendarReport’: Company’s calendar
- **fundamentalDataOptions** (*List[TagValue]*) – Unknown

**Return type**

*str*

**reqScannerData**(*subscription*, *scannerSubscriptionOptions*=[], *scannerSubscriptionFilterOptions*=[])

Do a blocking market scan by starting a subscription and canceling it after the initial list of results are in.

This method is blocking.

[https://interactivebrokers.github.io/tws-api/market\\_scanners.html](https://interactivebrokers.github.io/tws-api/market_scanners.html)

**Parameters**

- **subscription** (*ScannerSubscription*) – Basic filters.
- **scannerSubscriptionOptions** (*List[TagValue]*) – Unknown.
- **scannerSubscriptionFilterOptions** (*List[TagValue]*) – Advanced generic filters.

**Return type**

*ScanDataList*

**reqScannerSubscription**(*subscription*, *scannerSubscriptionOptions*=[],  
*scannerSubscriptionFilterOptions*=[])

Subscribe to market scan data.

[https://interactivebrokers.github.io/tws-api/market\\_scanners.html](https://interactivebrokers.github.io/tws-api/market_scanners.html)

**Parameters**

- **subscription** (*ScannerSubscription*) – What to scan for.
- **scannerSubscriptionOptions** (*List[TagValue]*) – Unknown.
- **scannerSubscriptionFilterOptions** (*List[TagValue]*) – Unknown.

**Return type**

*ScanDataList*

**cancelScannerSubscription(*dataList*)**

Cancel market data subscription.

[https://interactivebrokers.github.io/tws-api/market\\_scanners.html](https://interactivebrokers.github.io/tws-api/market_scanners.html)

**Parameters**

**dataList** (*ScanDataList*) – The scan data list that was obtained from *reqScannerSubscription()*.

**reqScannerParameters()**

Requests an XML list of scanner parameters.

This method is blocking.

**Return type**

*str*

**calculateImpliedVolatility(*contract*, *optionPrice*, *underPrice*, *implVolOptions*=[])**

Calculate the volatility given the option price.

This method is blocking.

[https://interactivebrokers.github.io/tws-api/option\\_computations.html](https://interactivebrokers.github.io/tws-api/option_computations.html)

**Parameters**

- **contract** (*Contract*) – Option contract.
- **optionPrice** (*float*) – Option price to use in calculation.
- **underPrice** (*float*) – Price of the underlier to use in calculation
- **implVolOptions** (*List[TagValue]*) – Unknown

**Return type**

*OptionComputation*

**calculateOptionPrice(*contract*, *volatility*, *underPrice*, *optPrcOptions*=[])**

Calculate the option price given the volatility.

This method is blocking.

[https://interactivebrokers.github.io/tws-api/option\\_computations.html](https://interactivebrokers.github.io/tws-api/option_computations.html)

**Parameters**

- **contract** (*Contract*) – Option contract.
- **volatility** (*float*) – Option volatility to use in calculation.
- **underPrice** (*float*) – Price of the underlier to use in calculation
- **implVolOptions** – Unknown

**Return type**

*OptionComputation*

**reqSecDefOptParams(*underlyingSymbol*, *futFopExchange*, *underlyingSecType*, *underlyingConId*)**

Get the option chain.

This method is blocking.

<https://interactivebrokers.github.io/tws-api/options.html>

**Parameters**

- **underlyingSymbol** (*str*) – Symbol of underlier contract.

- **futFopExchange** (*str*) – Exchange (only for FuturesOption, otherwise leave blank).
- **underlyingSecType** (*str*) – The type of the underlying security, like ‘STK’ or ‘FUT’.
- **underlyingConId** (*int*) – conId of the underlying contract.

**Return type***List[OptionChain]***exerciseOptions**(*contract, exerciseAction, exerciseQuantity, account, override*)

Exercise an options contract.

<https://interactivebrokers.github.io/tws-api/options.html>**Parameters**

- **contract** (*Contract*) – The option contract to be exercised.
- **exerciseAction** (*int*) –
  - 1 = exercise the option
  - 2 = let the option lapse
- **exerciseQuantity** (*int*) – Number of contracts to be exercised.
- **account** (*str*) – Destination account.
- **override** (*int*) –
  - 0 = no override
  - 1 = override the system’s natural action

**reqNewsProviders**()

Get a list of news providers.

This method is blocking.

**Return type***List[NewsProvider]***reqNewsArticle**(*providerCode, articleId, newsArticleOptions=[]*)

Get the body of a news article.

This method is blocking.

<https://interactivebrokers.github.io/tws-api/news.html>**Parameters**

- **providerCode** (*str*) – Code indicating news provider, like ‘BZ’ or ‘FLY’.
- **articleId** (*str*) – ID of the specific article.
- **newsArticleOptions** (*List[TagValue]*) – Unknown.

**Return type***NewsArticle***reqHistoricalNews**(*conId, providerCodes, startDateTime, endDateTime, totalResults, historicalNewsOptions=[]*)

Get historical news headline.

<https://interactivebrokers.github.io/tws-api/news.html>

This method is blocking.

**Parameters**

- **conId** (*int*) – Search news articles for contract with this conId.
- **providerCodes** (*str*) – A ‘+’-separated list of provider codes, like ‘BZ+FLY’.
- **startDateTime** (*Union[str, date]*) – The (exclusive) start of the date range. Can be given as a *datetime.date* or *datetime.datetime*, or it can be given as a string in ‘yyyymmdd HH:mm:ss’ format. If no timezone is given then the TWS login timezone is used.
- **endDateTime** (*Union[str, date]*) – The (inclusive) end of the date range. Can be given as a *datetime.date* or *datetime.datetime*, or it can be given as a string in ‘yyyymmdd HH:mm:ss’ format. If no timezone is given then the TWS login timezone is used.
- **totalResults** (*int*) – Maximum number of headlines to fetch (300 max).
- **historicalNewsOptions** (*List[TagValue]*) – Unknown.

**Return type***HistoricalNews***reqNewsBulletins**(*allMessages*)

Subscribe to IB news bulletins.

<https://interactivebrokers.github.io/tws-api/news.html>**Parameters**

- **allMessages** (*bool*) – If True then fetch all messages for the day.

**cancelNewsBulletins**()

Cancel subscription to IB news bulletins.

**requestFA**(*faDataType*)

Requests to change the FA configuration.

This method is blocking.

**Parameters**

- **faDataType** (*int*) –

- 1 = Groups: Offer traders a way to create a group of accounts and apply a single allocation method to all accounts in the group.
- 2 = Profiles: Let you allocate shares on an account-by-account basis using a predefined calculation value.
- 3 = Account Aliases: Let you easily identify the accounts by meaningful names rather than account numbers.

**replaceFA**(*faDataType, xml*)

Replaces Financial Advisor’s settings.

**Parameters**

- **faDataType** (*int*) – See [requestFA\(\)](#).
- **xml** (*str*) – The XML-formatted configuration string.

**reqWshMetaData**()

Request Wall Street Horizon metadata.

<https://interactivebrokers.github.io/tws-api/fundamentals.html>

**cancelWshMetaData()**

Cancel WSH metadata.

**reqWshEventData(data)**

Request Wall Street Horizon event data.

[`reqWshMetaData\(\)`](#) must have been called first before using this method.

**Parameters**

**data** ([`WshEventData`](#)) – Filters for selecting the corporate event data.

[https://interactivebrokers.github.io/tws-api/wshe\\_filters.html](https://interactivebrokers.github.io/tws-api/wshe_filters.html)

**cancelWshEventData()**

Cancel active WSH event data.

**getWshMetaData()**

Blocking convenience method that returns the WSH metadata (that is the available filters and event types) as a JSON string.

Please note that a [Wall Street Horizon](#) subscription is required.

```
# Get the list of available filters and event types:
meta = ib.getWshMetaData()
print(meta)
```

**Return type**

str

**getWshEventData(data)**

Blocking convenience method that returns the WSH event data as a JSON string. [`getWshMetaData\(\)`](#) must have been called first before using this method.

Please note that a [Wall Street Horizon](#) subscription is required.

```
# For IBM (with conId=8314) query the:
#   - Earnings Dates (wshe_ed)
#   - Board of Directors meetings (wshe_bod)
data = WshEventData(
    filter = '''{
        "country": "All",
        "watchlist": ["8314"],
        "limit_region": 10,
        "limit": 10,
        "wshe_ed": "true",
        "wshe_bod": "true"
    }'''
)
events = ib.getWshEventData(data)
print(events)
```

**Return type**

str

**reqUserInfo()**

Get the White Branding ID of the user.

**Return type**`str`

**async connectAsync**(*host*='127.0.0.1', *port*=7497, *clientId*=1, *timeout*=4, *readonly*=False, *account*="", *raiseSyncErrors*=False)

**async qualifyContractsAsync**(\**contracts*)

**Return type**`List[Contract]`

**async reqTickersAsync**(\**contracts*, *regulatorySnapshot*=False)

**Return type**`List[Ticker]`

**whatIfOrderAsync**(*contract*, *order*)

**Return type**`Awaitable[OrderState]`

**reqCurrentTimeAsync**()

**Return type**`Awaitable[datetime]`

**reqAccountUpdatesAsync**(*account*)

**Return type**`Awaitable[None]`

**reqAccountUpdatesMultiAsync**(*account*, *modelCode*="")

**Return type**`Awaitable[None]`

**async accountSummaryAsync**(*account*="")

**Return type**`List[AccountValue]`

**reqAccountSummaryAsync**()

**Return type**`Awaitable[None]`

**reqOpenOrdersAsync**()

**Return type**`Awaitable[List[Trade]]`

**reqAllOpenOrdersAsync**()

**Return type**`Awaitable[List[Trade]]`

**reqCompletedOrdersAsync**(*apiOnly*)

**Return type**`Awaitable[List[Trade]]`

**reqExecutionsAsync**(*execFilter=None*)

Return type

`Awaitable[List[Fill]]`

**reqPositionsAsync**()

Return type

`Awaitable[List[Position]]`

**reqContractDetailsAsync**(*contract*)

Return type

`Awaitable[List[ContractDetails]]`

**async reqMatchingSymbolsAsync**(*pattern*)

Return type

`Optional[List[ContractDescription]]`

**async reqMarketRuleAsync**(*marketRuleId*)

Return type

`Optional[List[PriceIncrement]]`

**async reqHistoricalDataAsync**(*contract, endDateTime, durationStr, barSizeSetting, whatToShow, useRTH, formatDate=1, keepUpToDate=False, chartOptions=[], timeout=60*)

Return type

`BarDataList`

**reqHistoricalScheduleAsync**(*contract, numDays, endDateTime="", useRTH=True*)

Return type

`Awaitable[HistoricalSchedule]`

**reqHistoricalTicksAsync**(*contract, startDateTime, endDateTime, numberOfTicks, whatToShow, useRth, ignoreSize=False, miscOptions=[]*)

Return type

`Awaitable[List]`

**async reqHeadTimeStampAsync**(*contract, whatToShow, useRTH, formatDate*)

Return type

`datetime`

**reqSmartComponentsAsync**(*bboExchange*)

**reqMktDepthExchangesAsync**()

Return type

`Awaitable[List[DepthMktDataDescription]]`

**reqHistogramDataAsync**(*contract, useRTH, period*)

Return type

`Awaitable[List[HistogramData]]`



**reqFundamentalDataAsync**(*contract*, *reportType*, *fundamentalDataOptions*=[])

Return type

*Awaitable*[*str*]

**async reqScannerDataAsync**(*subscription*, *scannerSubscriptionOptions*=[],  
*scannerSubscriptionFilterOptions*=[])

Return type

*ScanDataList*

**reqScannerParametersAsync**()

Return type

*Awaitable*[*str*]

**async calculateImpliedVolatilityAsync**(*contract*, *optionPrice*, *underPrice*, *implVolOptions*=[])

Return type

*Optional*[*OptionComputation*]

**async calculateOptionPriceAsync**(*contract*, *volatility*, *underPrice*, *optPrcOptions*=[])

Return type

*Optional*[*OptionComputation*]

**reqSecDefOptParamsAsync**(*underlyingSymbol*, *futFopExchange*, *underlyingSecType*, *underlyingConId*)

Return type

*Awaitable*[*List*[*OptionChain*]]

**reqNewsProvidersAsync**()

Return type

*Awaitable*[*List*[*NewsProvider*]]

**reqNewsArticleAsync**(*providerCode*, *articleId*, *newsArticleOptions*=[])

Return type

*Awaitable*[*NewsArticle*]

**async reqHistoricalNewsAsync**(*conId*, *providerCodes*, *startDateTime*, *endDateTime*, *totalResults*,  
*historicalNewsOptions*=[])

Return type

*Optional*[*HistoricalNews*]

**async requestFAAsync**(*faDataType*)

**async getWshMetaDataAsync**()

Return type

*str*

**async getWshEventDataAsync**(*data*)

Return type

*str*

**reqUserInfoAsync**()

## 2.2 Client

Socket client for communicating with Interactive Brokers.

**class** `ib_insync.client.Client(wrapper)`

Replacement for `ibapi.client.EClient` that uses asyncio.

The client is fully asynchronous and has its own event-driven networking code that replaces the networking code of the standard `EClient`. It also replaces the infinite loop of `EClient.run()` with the asyncio event loop. It can be used as a drop-in replacement for the standard `EClient` as provided by IBAPI.

Compared to the standard `EClient` this client has the following additional features:

- `client.connect()` will block until the client is ready to serve requests; It is not necessary to wait for `nextValidId` to start requests as the client has already done that. The `reqId` is directly available with `getReqId()`.
- `client.connectAsync()` is a coroutine for connecting asynchronously.
- When blocking, `client.connect()` can be made to time out with the `timeout` parameter (default 2 seconds).
- Optional `wrapper.priceSizeTick(reqId, tickType, price, size)` that combines price and size instead of the two wrapper methods `priceTick` and `sizeTick`.
- Automatic request throttling.
- Optional `wrapper.tcpDataArrived()` method; If the wrapper has this method it is invoked directly after a network packet has arrived. A possible use is to timestamp all data in the packet with the exact same time.
- Optional `wrapper.tcpDataProcessed()` method; If the wrapper has this method it is invoked after the network packet's data has been handled. A possible use is to write or evaluate the newly arrived data in one batch instead of item by item.

### Parameters

- **MaxRequests** (*int*) – Throttle the number of requests to `MaxRequests` per `RequestsInterval` seconds. Set to 0 to disable throttling.
- **RequestsInterval** (*float*) – Time interval (in seconds) for request throttling.
- **MinClientVersion** (*int*) – Client protocol version.
- **MaxClientVersion** (*int*) – Client protocol version.

### Events:

- `apiStart()`
- `apiEnd()`
- `apiError(errorMsg: str)`
- `throttleStart()`
- `throttleEnd()`

```
events = ('apiStart', 'apiEnd', 'apiError', 'throttleStart', 'throttleEnd')
```

```
MaxRequests = 45
```

```
RequestsInterval = 1
```

**MinClientVersion** = 157

**MaxClientVersion** = 178

**DISCONNECTED** = 0

**CONNECTING** = 1

**CONNECTED** = 2

**reset()**

**serverVersion()**

**Return type**

`int`

**run()**

**isConnected()**

**isReady()**

Is the API connection up and running?

**Return type**

`bool`

**connectionStats()**

Get statistics about the connection.

**Return type**

`ConnectionStats`

**getReqId()**

Get new request ID.

**Return type**

`int`

**updateReqId(*minReqId*)**

Update the next reqId to be at least minReqId.

**getAccounts()**

Get the list of account names that are under management.

**Return type**

`List[str]`

**setConnectOptions(*connectOptions*)**

Set additional connect options.

**Parameters**

**connectOptions** (`str`) – Use “+PACEAPI” to use request-pacing built into TWS/gateway 974+ (obsolete).

**connect(*host, port, clientId, timeout=2.0*)**

Connect to a running TWS or IB gateway application.

**Parameters**

- **host** (`str`) – Host name or IP address.

- **port** (*int*) – Port number.
- **clientId** (*int*) – ID number to use for this client; must be unique per connection.
- **timeout** (*Optional[float]*) – If establishing the connection takes longer than **timeout** seconds then the `asyncio.TimeoutError` exception is raised. Set to 0 to disable timeout.

**async connectAsync**(*host, port, clientId, timeout=2.0*)

**disconnect**()

Disconnect from IB connection.

**send**(*\*fields, makeEmpty=True*)

Serialize and send the given fields using the IB socket protocol.

**sendMsg**(*msg*)

**reqMktData**(*reqId, contract, genericTickList, snapshot, regulatorySnapshot, mktDataOptions*)

**cancelMktData**(*reqId*)

**placeOrder**(*orderId, contract, order*)

**cancelOrder**(*orderId, manualCancelOrderTime=""*)

**reqOpenOrders**()

**reqAccountUpdates**(*subscribe, acctCode*)

**reqExecutions**(*reqId, execFilter*)

**reqIds**(*numIds*)

**reqContractDetails**(*reqId, contract*)

**reqMktDepth**(*reqId, contract, numRows, isSmartDepth, mktDepthOptions*)

**cancelMktDepth**(*reqId, isSmartDepth*)

**reqNewsBulletins**(*allMsgs*)

**cancelNewsBulletins**()

**setServerLogLevel**(*logLevel*)

**reqAutoOpenOrders**(*bAutoBind*)

**reqAllOpenOrders**()

**reqManagedAccts**()

**requestFA**(*faData*)

**replaceFA**(*reqId, faData, cxml*)

**reqHistoricalData**(*reqId, contract, endDateTime, durationStr, barSizeSetting, whatToShow, useRTH, formatDate, keepUpToDate, chartOptions*)

**exerciseOptions**(*reqId, contract, exerciseAction, exerciseQuantity, account, override*)

---

```
reqScannerSubscription(reqId, subscription, scannerSubscriptionOptions,  
                        scannerSubscriptionFilterOptions)  
  
cancelScannerSubscription(reqId)  
  
reqScannerParameters()  
  
cancelHistoricalData(reqId)  
  
reqCurrentTime()  
  
reqRealTimeBars(reqId, contract, barSize, whatToShow, useRTH, realTimeBarsOptions)  
  
cancelRealTimeBars(reqId)  
  
reqFundamentalData(reqId, contract, reportType, fundamentalDataOptions)  
  
cancelFundamentalData(reqId)  
  
calculateImpliedVolatility(reqId, contract, optionPrice, underPrice, implVolOptions)  
  
calculateOptionPrice(reqId, contract, volatility, underPrice, optPrcOptions)  
  
cancelCalculateImpliedVolatility(reqId)  
  
cancelCalculateOptionPrice(reqId)  
  
reqGlobalCancel()  
  
reqMarketDataType(marketDataType)  
  
reqPositions()  
  
reqAccountSummary(reqId, groupName, tags)  
  
cancelAccountSummary(reqId)  
  
cancelPositions()  
  
verifyRequest(apiName, apiVersion)  
  
verifyMessage(apiData)  
  
queryDisplayGroups(reqId)  
  
subscribeToGroupEvents(reqId, groupId)  
  
updateDisplayGroup(reqId, contractInfo)  
  
unsubscribeFromGroupEvents(reqId)  
  
startApi()  
  
verifyAndAuthRequest(apiName, apiVersion, opaqueIsoKey)  
  
verifyAndAuthMessage(apiData, xyzResponse)  
  
reqPositionsMulti(reqId, account, modelCode)  
  
cancelPositionsMulti(reqId)
```

**reqAccountUpdatesMulti**(*reqId, account, modelCode, ledgerAndNLV*)

**cancelAccountUpdatesMulti**(*reqId*)

**reqSecDefOptParams**(*reqId, underlyingSymbol, futFopExchange, underlyingSecType, underlyingConId*)

**reqSoftDollarTiers**(*reqId*)

**reqFamilyCodes**()

**reqMatchingSymbols**(*reqId, pattern*)

**reqMktDepthExchanges**()

**reqSmartComponents**(*reqId, bboExchange*)

**reqNewsArticle**(*reqId, providerCode, articleId, newsArticleOptions*)

**reqNewsProviders**()

**reqHistoricalNews**(*reqId, conId, providerCodes, startDateTime, endDateTime, totalResults, historicalNewsOptions*)

**reqHeadTimeStamp**(*reqId, contract, whatToShow, useRTH, formatDate*)

**reqHistogramData**(*tickerId, contract, useRTH, timePeriod*)

**cancelHistogramData**(*tickerId*)

**cancelHeadTimeStamp**(*reqId*)

**reqMarketRule**(*marketRuleId*)

**reqPnL**(*reqId, account, modelCode*)

**cancelPnL**(*reqId*)

**reqPnLSingle**(*reqId, account, modelCode, conid*)

**cancelPnLSingle**(*reqId*)

**reqHistoricalTicks**(*reqId, contract, startDateTime, endDateTime, numberOfTicks, whatToShow, useRth, ignoreSize, miscOptions*)

**reqTickByTickData**(*reqId, contract, tickType, numberOfTicks, ignoreSize*)

**cancelTickByTickData**(*reqId*)

**reqCompletedOrders**(*apiOnly*)

**reqWshMetaData**(*reqId*)

**cancelWshMetaData**(*reqId*)

**reqWshEventData**(*reqId, data*)

**cancelWshEventData**(*reqId*)

**reqUserInfo**(*reqId*)

## 2.3 Order

Order types used by Interactive Brokers.

```

class ib_insync.order.Order(orderId=0, clientId=0, permId=0, action="", totalQuantity=0.0, orderType="",
    lmtPrice=1.7976931348623157e+308, auxPrice=1.7976931348623157e+308,
    tif="", activeStartTime="", activeStopTime="", ocaGroup="", ocaType=0,
    orderRef="", transmit=True, parentId=0, blockOrder=False, sweepToFill=False,
    displaySize=0, triggerMethod=0, outsideRth=False, hidden=False,
    goodAfterTime="", goodTillDate="", rule80A="", allOrNone=False,
    minQty=2147483647, percentOffset=1.7976931348623157e+308,
    overridePercentageConstraints=False,
    trailStopPrice=1.7976931348623157e+308,
    trailingPercent=1.7976931348623157e+308, faGroup="", faProfile="",
    faMethod="", faPercentage="", designatedLocation="", openClose='O', origin=0,
    shortSaleSlot=0, exemptCode=-1, discretionaryAmt=0.0, eTradeOnly=False,
    firmQuoteOnly=False, nbboPriceCap=1.7976931348623157e+308,
    optOutSmartRouting=False, auctionStrategy=0,
    startingPrice=1.7976931348623157e+308,
    stockRefPrice=1.7976931348623157e+308,
    delta=1.7976931348623157e+308,
    stockRangeLower=1.7976931348623157e+308,
    stockRangeUpper=1.7976931348623157e+308, randomizePrice=False,
    randomizeSize=False, volatility=1.7976931348623157e+308,
    volatilityType=2147483647, deltaNeutralOrderType="",
    deltaNeutralAuxPrice=1.7976931348623157e+308, deltaNeutralConId=0,
    deltaNeutralSettlingFirm="", deltaNeutralClearingAccount="",
    deltaNeutralClearingIntent="", deltaNeutralOpenClose="",
    deltaNeutralShortSale=False, deltaNeutralShortSaleSlot=0,
    deltaNeutralDesignatedLocation="", continuousUpdate=False,
    referencePriceType=2147483647, basisPoints=1.7976931348623157e+308,
    basisPointsType=2147483647, scaleInitLevelSize=2147483647,
    scaleSubsLevelSize=2147483647,
    scalePriceIncrement=1.7976931348623157e+308,
    scalePriceAdjustValue=1.7976931348623157e+308,
    scalePriceAdjustInterval=2147483647,
    scaleProfitOffset=1.7976931348623157e+308, scaleAutoReset=False,
    scaleInitPosition=2147483647, scaleInitFillQty=2147483647,
    scaleRandomPercent=False, scaleTable="", hedgeType="", hedgeParam="",
    account="", settlingFirm="", clearingAccount="", clearingIntent="",
    algoStrategy="", algoParams=<factory>,
    smartComboRoutingParams=<factory>, algoId="", whatIf=False,
    notHeld=False, solicited=False, modelCode="", orderComboLegs=<factory>,
    orderMiscOptions=<factory>, referenceContractId=0,
    peggedChangeAmount=0.0, isPeggedChangeAmountDecrease=False,
    referenceChangeAmount=0.0, referenceExchangeId="", adjustedOrderType="",
    triggerPrice=1.7976931348623157e+308,
    adjustedStopPrice=1.7976931348623157e+308,
    adjustedStopLimitPrice=1.7976931348623157e+308,
    adjustedTrailingAmount=1.7976931348623157e+308,
    adjustableTrailingUnit=0, lmtPriceOffset=1.7976931348623157e+308,
    conditions=<factory>, conditionsCancelOrder=False,
    conditionsIgnoreRth=False, extOperator="", softDollarTier=<factory>,
    cashQty=1.7976931348623157e+308, mifid2DecisionMaker="",
    mifid2DecisionAlgo="", mifid2ExecutionTrader="", mifid2ExecutionAlgo="",
    dontUseAutoPriceForHedge=False, isOmsContainer=False,
    discretionaryUpToLimitPrice=False, autoCancelDate="",
    filledQuantity=1.7976931348623157e+308, refFuturesConId=0,
    autoCancelParent=False, shareholder="", imbalanceOnly=False,
    routeMarketableToBbo=False, parentPermId=0, usePriceMgmtAlgo=False,
    duration=2147483647, postToAts=2147483647, advancedOrderPermId=0,
    manualOrderTime="", minTradeQty=2147483647,
    minCompeteSize=2147483647,
    competeAgainstBestOffset=1.7976931348623157e+308,

```



Order for trading contracts.

[https://interactivebrokers.github.io/tws-api/available\\_orders.html](https://interactivebrokers.github.io/tws-api/available_orders.html)

```
orderId:  int = 0
clientId:  int = 0
permId:   int = 0
action:   str = ''
totalQuantity: float = 0.0
orderType: str = ''
lmtPrice: float = 1.7976931348623157e+308
auxPrice: float = 1.7976931348623157e+308
tif:      str = ''
activeStartTime: str = ''
activeStopTime: str = ''
ocaGroup: str = ''
ocaType:  int = 0
orderRef: str = ''
transmit: bool = True
parentId:  int = 0
blockOrder: bool = False
sweepToFill: bool = False
displaySize: int = 0
triggerMethod: int = 0
outsideRth: bool = False
hidden:     bool = False
goodAfterTime: str = ''
goodTillDate: str = ''
rule80A:    str = ''
allOrNone:  bool = False
minQty:     int = 2147483647
percentOffset: float = 1.7976931348623157e+308
```

```
overridePercentageConstraints: bool = False
trailStopPrice: float = 1.7976931348623157e+308
trailingPercent: float = 1.7976931348623157e+308
faGroup: str = ''
faProfile: str = ''
faMethod: str = ''
faPercentage: str = ''
designatedLocation: str = ''
openClose: str = 'O'
origin: int = 0
shortSaleSlot: int = 0
exemptCode: int = -1
discretionaryAmt: float = 0.0
eTradeOnly: bool = False
firmQuoteOnly: bool = False
nbboPriceCap: float = 1.7976931348623157e+308
optOutSmartRouting: bool = False
auctionStrategy: int = 0
startingPrice: float = 1.7976931348623157e+308
stockRefPrice: float = 1.7976931348623157e+308
delta: float = 1.7976931348623157e+308
stockRangeLower: float = 1.7976931348623157e+308
stockRangeUpper: float = 1.7976931348623157e+308
randomizePrice: bool = False
randomizeSize: bool = False
volatility: float = 1.7976931348623157e+308
volatilityType: int = 2147483647
deltaNeutralOrderType: str = ''
deltaNeutralAuxPrice: float = 1.7976931348623157e+308
deltaNeutralConId: int = 0
deltaNeutralSettlingFirm: str = ''
```

```
deltaNeutralClearingAccount: str = ''
deltaNeutralClearingIntent: str = ''
deltaNeutralOpenClose: str = ''
deltaNeutralShortSale: bool = False
deltaNeutralShortSaleSlot: int = 0
deltaNeutralDesignatedLocation: str = ''
continuousUpdate: bool = False
referencePriceType: int = 2147483647
basisPoints: float = 1.7976931348623157e+308
basisPointsType: int = 2147483647
scaleInitLevelSize: int = 2147483647
scaleSubsLevelSize: int = 2147483647
scalePriceIncrement: float = 1.7976931348623157e+308
scalePriceAdjustValue: float = 1.7976931348623157e+308
scalePriceAdjustInterval: int = 2147483647
scaleProfitOffset: float = 1.7976931348623157e+308
scaleAutoReset: bool = False
scaleInitPosition: int = 2147483647
scaleInitFillQty: int = 2147483647
scaleRandomPercent: bool = False
scaleTable: str = ''
hedgeType: str = ''
hedgeParam: str = ''
account: str = ''
settlingFirm: str = ''
clearingAccount: str = ''
clearingIntent: str = ''
algoStrategy: str = ''
algoParams: List[TagValue]
smartComboRoutingParams: List[TagValue]
algoId: str = ''
```

```
whatIf: bool = False
notHeld: bool = False
solicited: bool = False
modelCode: str = ''
orderComboLegs: List[OrderComboLeg]
orderMiscOptions: List[TagValue]
referenceContractId: int = 0
peggedChangeAmount: float = 0.0
isPeggedChangeAmountDecrease: bool = False
referenceChangeAmount: float = 0.0
referenceExchangeId: str = ''
adjustedOrderType: str = ''
triggerPrice: float = 1.7976931348623157e+308
adjustedStopPrice: float = 1.7976931348623157e+308
adjustedStopLimitPrice: float = 1.7976931348623157e+308
adjustedTrailingAmount: float = 1.7976931348623157e+308
adjustableTrailingUnit: int = 0
lmtPriceOffset: float = 1.7976931348623157e+308
conditions: List[OrderCondition]
conditionsCancelOrder: bool = False
conditionsIgnoreRth: bool = False
extOperator: str = ''
softDollarTier: SoftDollarTier
cashQty: float = 1.7976931348623157e+308
mifid2DecisionMaker: str = ''
mifid2DecisionAlgo: str = ''
mifid2ExecutionTrader: str = ''
mifid2ExecutionAlgo: str = ''
dontUseAutoPriceForHedge: bool = False
isOmsContainer: bool = False
discretionaryUpToLimitPrice: bool = False
```

```

autoCancelDate:  str = ''
filledQuantity:  float = 1.7976931348623157e+308
refFuturesConId:  int = 0
autoCancelParent:  bool = False
shareholder:  str = ''
imbalanceOnly:  bool = False
routeMarketableToBbo:  bool = False
parentPermId:  int = 0
usePriceMgmtAlgo:  bool = False
duration:  int = 2147483647
postToAths:  int = 2147483647
advancedErrorOverride:  str = ''
manualOrderTime:  str = ''
minTradeQty:  int = 2147483647
minCompeteSize:  int = 2147483647
competeAgainstBestOffset:  float = 1.7976931348623157e+308
midOffsetAtWhole:  float = 1.7976931348623157e+308
midOffsetAtHalf:  float = 1.7976931348623157e+308

```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.order.LimitOrder`(*action*, *totalQuantity*, *lmtPrice*, *\*\*kwargs*)

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update**(*\*srcObjs*, *\*\*kwargs*)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.order.MarketOrder`(*action*, *totalQuantity*, *\*\*kwargs*)

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update**(*\*srcObjs*, *\*\*kwargs*)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.order.StopOrder`(*action*, *totalQuantity*, *stopPrice*, *\*\*kwargs*)

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.order.StopLimitOrder`(*action*, *totalQuantity*, *lmtPrice*, *stopPrice*, *\*\*kwargs*)

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.order.OrderStatus`(*orderId=0*, *status=""*, *filled=0.0*, *remaining=0.0*, *avgFillPrice=0.0*, *permId=0*, *parentId=0*, *lastFillPrice=0.0*, *clientId=0*, *whyHeld=""*, *mktCapPrice=0.0*)

**orderId:** `int` = 0

**status:** `str` = ''

**filled:** `float` = 0.0

**remaining:** `float` = 0.0

**avgFillPrice:** `float` = 0.0

```
permId: int = 0
parentId: int = 0
lastFillPrice: float = 0.0
clientId: int = 0
whyHeld: str = ''
mktCapPrice: float = 0.0
PendingSubmit: ClassVar[str] = 'PendingSubmit'
PendingCancel: ClassVar[str] = 'PendingCancel'
PreSubmitted: ClassVar[str] = 'PreSubmitted'
Submitted: ClassVar[str] = 'Submitted'
ApiPending: ClassVar[str] = 'ApiPending'
ApiCancelled: ClassVar[str] = 'ApiCancelled'
Cancelled: ClassVar[str] = 'Cancelled'
Filled: ClassVar[str] = 'Filled'
Inactive: ClassVar[str] = 'Inactive'
DoneStates: ClassVar[FrozenSet[str]] = frozenset({'ApiCancelled', 'Cancelled',
'Filled'})
ActiveStates: ClassVar[FrozenSet[str]] = frozenset({'ApiPending', 'PendingSubmit',
'PreSubmitted', 'Submitted'})
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**  
*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**  
*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**  
*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**  
*object*



```
class ib_insync.order.OrderState(status="", initMarginBefore="", maintMarginBefore="",
                                equityWithLoanBefore="", initMarginChange="", maintMarginChange="",
                                equityWithLoanChange="", initMarginAfter="", maintMarginAfter="",
                                equityWithLoanAfter="", commission=1.7976931348623157e+308,
                                minCommission=1.7976931348623157e+308,
                                maxCommission=1.7976931348623157e+308, commissionCurrency="",
                                warningText="", completedTime="", completedStatus="")
```

```
status: str = ''
initMarginBefore: str = ''
maintMarginBefore: str = ''
equityWithLoanBefore: str = ''
initMarginChange: str = ''
maintMarginChange: str = ''
equityWithLoanChange: str = ''
initMarginAfter: str = ''
maintMarginAfter: str = ''
equityWithLoanAfter: str = ''
commission: float = 1.7976931348623157e+308
minCommission: float = 1.7976931348623157e+308
maxCommission: float = 1.7976931348623157e+308
commissionCurrency: str = ''
warningText: str = ''
completedTime: str = ''
completedStatus: str = ''
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** ib\_insync.order.**OrderComboLeg**(price=1.7976931348623157e+308)

**price:** *float* = 1.7976931348623157e+308

**dict**()

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults**()

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple**()

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** ib\_insync.order.**Trade**(contract=<factory>, order=<factory>, orderStatus=<factory>,  
fills=<factory>, log=<factory>, advancedError="")

Trade keeps track of an order, its status and all its fills.

**Events:**

- `statusEvent` (trade: *Trade*)
- `modifyEvent` (trade: *Trade*)
- `fillEvent` (trade: *Trade*, fill: *Fill*)
- `commissionReportEvent` (trade: *Trade*, fill: *Fill*, commissionReport: *CommissionReport*)
- `filledEvent` (trade: *Trade*)
- `cancelEvent` (trade: *Trade*)
- `cancelledEvent` (trade: *Trade*)

**contract:** *Contract*

**order:** *Order*

**orderStatus:** *OrderStatus*

**fills:** `List[Fill]`

**log:** `List[TradeLogEntry]`

**advancedError:** `str = ''`

**events:** `ClassVar = ('statusEvent', 'modifyEvent', 'fillEvent', 'commissionReportEvent', 'filledEvent', 'cancelEvent', 'cancelledEvent')`

**isActive()**

True if eligible for execution, false otherwise.

**Return type**

`bool`

**isDone()**

True if completely filled or cancelled, false otherwise.

**Return type**

`bool`

**filled()**

Number of shares filled.

**Return type**

`float`

**remaining()**

Number of shares remaining to be filled.

**Return type**

`float`

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

`dict`

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

**class** `ib_insync.order.BasketOrder(parent, takeProfit, stopLoss)`

Create new instance of `BasketOrder(parent, takeProfit, stopLoss)`

parent: *Order*

takeProfit: *Order*

stopLoss: *Order*

**class** ib\_insync.order.OrderCondition

**static** createClass(*condType*)

**And**()

**Or**()

**dict**()

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults**()

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple**()

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update**(\**srcObjs*, \*\**kwargs*)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** ib\_insync.order.PriceCondition(*condType=1*, *conjunction='a'*, *isMore=True*, *price=0.0*, *conId=0*,  
*exch=""*, *triggerMethod=0*)

condType: *int* = 1

conjunction: *str* = 'a'

isMore: *bool* = True

price: *float* = 0.0

conId: *int* = 0

exch: *str* = ''

triggerMethod: *int* = 0

**dict**()

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.order.TimeCondition(condType=3, conjunction='a', isMore=True, time='')

```

```
condType: int = 3

```

```
conjunction: str = 'a'

```

```
isMore: bool = True

```

```
time: str = ''

```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.order.MarginCondition(condType=4, conjunction='a', isMore=True, percent=0)

```

```
condType: int = 4

```

```
conjunction: str = 'a'

```

**isMore:** `bool` = `True`

**percent:** `int` = `0`

**dict()**

Return dataclass values as `dict`. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**  
*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as `dict`.

**Return type**  
*dict*

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**  
*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**  
*object*

**class** `ib_insync.order.ExecutionCondition`(*condType=5, conjunction='a', secType='', exch='', symbol=''*)

**condType:** `int` = `5`

**conjunction:** `str` = `'a'`

**secType:** `str` = `''`

**exch:** `str` = `''`

**symbol:** `str` = `''`

**dict()**

Return dataclass values as `dict`. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**  
*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as `dict`.

**Return type**  
*dict*

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**  
*tuple*

**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.order.VolumeCondition(condType=6, conjunction='a', isMore=True, volume=0, conId=0,
                                     exch="")
```

**condType:** *int* = 6

**conjunction:** *str* = 'a'

**isMore:** *bool* = True

**volume:** *int* = 0

**conId:** *int* = 0

**exch:** *str* = ''

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.order.PercentChangeCondition(condType=7, conjunction='a', isMore=True,
                                             changePercent=0.0, conId=0, exch="")
```

**condType:** *int* = 7

**conjunction:** *str* = 'a'

**isMore:** *bool* = True

**changePercent:** *float* = 0.0

**conId:** *int* = 0

**exch:** `str` = ''

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

`dict`

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

## 2.4 Contract

Financial instrument types used by Interactive Brokers.

```
class ib_insync.contract.Contract(secType="", conId=0, symbol="", lastTradeDateOrContractMonth="",
                                  strike=0.0, right="", multiplier="", exchange="", primaryExchange="",
                                  currency="", localSymbol="", tradingClass="", includeExpired=False,
                                  secIdType="", secId="", description="", issuerId="",
                                  comboLegsDescrip="", comboLegs=<factory>,
                                  deltaNeutralContract=None)
```

`Contract(**kwargs)` can create any contract using keyword arguments. To simplify working with contracts, there are also more specialized contracts that take optional positional arguments. Some examples:

```
Contract(conId=270639)
Stock('AMD', 'SMART', 'USD')
Stock('INTC', 'SMART', 'USD', primaryExchange='NASDAQ')
Forex('EURUSD')
CFD('IBUS30')
Future('ES', '20180921', 'GLOBEX')
Option('SPY', '20170721', 240, 'C', 'SMART')
Bond(secIdType='ISIN', secId='US03076KAA60')
Crypto('BTC', 'PAXOS', 'USD')
```

### Parameters

- **conId** (`int`) – The unique IB contract identifier.
- **symbol** (`str`) – The contract (or its underlying) symbol.
- **secType** (`str`) – The security type:



- 'STK' = Stock (or ETF)
  - 'OPT' = Option
  - 'FUT' = Future
  - 'IND' = Index
  - 'FOP' = Futures option
  - 'CASH' = Forex pair
  - 'CFD' = CFD
  - 'BAG' = Combo
  - 'WAR' = Warrant
  - 'BOND' = Bond
  - 'CMDTY' = Commodity
  - 'NEWS' = News
  - 'FUND' = Mutual fund
  - 'CRYPTO' = Crypto currency
  - 'EVENT' = Bet on an event
- **lastTradeDateOrContractMonth** (*str*) – The contract's last trading day or contract month (for Options and Futures). Strings with format YYYYMM will be interpreted as the Contract Month whereas YYYYMMDD will be interpreted as Last Trading Day.
  - **strike** (*float*) – The option's strike price.
  - **right** (*str*) – Put or Call. Valid values are 'P', 'PUT', 'C', 'CALL', or '' for non-options.
  - **multiplier** (*str*) – The instrument's multiplier (i.e. options, futures).
  - **exchange** (*str*) – The destination exchange.
  - **currency** (*str*) – The underlying's currency.
  - **localSymbol** (*str*) – The contract's symbol within its primary exchange. For options, this will be the OCC symbol.
  - **primaryExchange** (*str*) – The contract's primary exchange. For smart routed contracts, used to define contract in case of ambiguity. Should be defined as native exchange of contract, e.g. ISLAND for MSFT. For exchanges which contain a period in name, will only be part of exchange name prior to period, i.e. ENEXT for ENEXT.BE.
  - **tradingClass** (*str*) – The trading class name for this contract. Available in TWS contract description window as well. For example, GBL Dec '13 future's trading class is "FGBL".
  - **includeExpired** (*bool*) – If set to true, contract details requests and historical data queries can be performed pertaining to expired futures contracts. Expired options or other instrument types are not available.
  - **secIdType** (*str*) – Security identifier type. Examples for Apple:
    - secIdType='ISIN', secId='US0378331005'
    - secIdType='CUSIP', secId='037833100'
  - **secId** (*str*) – Security identifier.
  - **comboLegsDescription** (*str*) – Description of the combo legs.

- **comboLegs** (*List[ComboLeg]*) – The legs of a combined contract definition.
- **deltaNeutralContract** (*DeltaNeutralContract*) – Delta and underlying price for Delta-Neutral combo orders.

```
secType: str = ''
conId: int = 0
symbol: str = ''
lastTradeDateOrContractMonth: str = ''
strike: float = 0.0
right: str = ''
multiplier: str = ''
exchange: str = ''
primaryExchange: str = ''
currency: str = ''
localSymbol: str = ''
tradingClass: str = ''
includeExpired: bool = False
secIdType: str = ''
secId: str = ''
description: str = ''
issuerId: str = ''
comboLegsDescrip: str = ''
comboLegs: List[ComboLeg]
deltaNeutralContract: Optional[DeltaNeutralContract] = None
```

**static create(\*\*kwargs)**

Create and return a specialized contract based on the given secType, or a general Contract if secType is not given.

**Return type**  
*Contract*

**isHashable()**

See if this contract can be hashed by conId.

Note: Bag contracts always get conId=28812380, so they're not hashable.

**Return type**  
*bool*

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.contract.Stock(symbol="", exchange="", currency="", **kwargs)`

Stock contract.

**Parameters**

- **symbol** (*str*) – Symbol name.
- **exchange** (*str*) – Destination exchange.
- **currency** (*str*) – Underlying currency.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.contract.Option(symbol="", lastTradeDateOrContractMonth="", strike=0.0, right="",
                                exchange="", multiplier="", currency="", **kwargs)
```

Option contract.

#### Parameters

- **symbol** (*str*) – Symbol name.
- **lastTradeDateOrContractMonth** (*str*) – The option's last trading day or contract month.
  - YYYYMM format: To specify last month
  - YYYYMMDD format: To specify last trading day
- **strike** (*float*) – The option's strike price.
- **right** (*str*) – Put or call option. Valid values are 'P', 'PUT', 'C' or 'CALL'.
- **exchange** (*str*) – Destination exchange.
- **multiplier** (*str*) – The contract multiplier.
- **currency** (*str*) – Underlying currency.

#### dict()

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

#### Return type

*dict*

#### nonDefaults()

For a dataclass instance get the fields that are different from the default values and return as dict.

#### Return type

*dict*

#### tuple()

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

#### Return type

*tuple*

#### update(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

#### Return type

*object*

```
class ib_insync.contract.Future(symbol="", lastTradeDateOrContractMonth="", exchange="",
                                localSymbol="", multiplier="", currency="", **kwargs)
```

Future contract.

#### Parameters

- **symbol** (*str*) – Symbol name.
- **lastTradeDateOrContractMonth** (*str*) – The option's last trading day or contract month.
  - YYYYMM format: To specify last month
  - YYYYMMDD format: To specify last trading day
- **exchange** (*str*) – Destination exchange.

- **localSymbol** (*str*) – The contract's symbol within its primary exchange.
- **multiplier** (*str*) – The contract multiplier.
- **currency** (*str*) – Underlying currency.

**dict()**

Return dataclass values as `dict`. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as `dict`.

**Return type**

*dict*

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.contract.ContFuture`(*symbol=""*, *exchange=""*, *localSymbol=""*, *multiplier=""*, *currency=""*, *\*\*kwargs*)

Continuous future contract.

**Parameters**

- **symbol** (*str*) – Symbol name.
- **exchange** (*str*) – Destination exchange.
- **localSymbol** (*str*) – The contract's symbol within its primary exchange.
- **multiplier** (*str*) – The contract multiplier.
- **currency** (*str*) – Underlying currency.

**dict()**

Return dataclass values as `dict`. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as `dict`.

**Return type**

*dict*

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

**class** `ib_insync.contract.Forex`(pair="", exchange='IDEALPRO', symbol="", currency="", \*\*kwargs)

Foreign exchange currency pair.

**Parameters**

- **pair** (`str`) – Shortcut for specifying symbol and currency, like 'EURUSD'.
- **exchange** (`str`) – Destination exchange.
- **symbol** (`str`) – Base currency.
- **currency** (`str`) – Quote currency.

**pair**()

Short name of pair.

**Return type**

`str`

**dict**()

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults**()

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

`dict`

**tuple**()

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

**class** `ib_insync.contract.Index`(symbol="", exchange="", currency="", \*\*kwargs)

Index.

**Parameters**

- **symbol** (`str`) – Symbol name.
- **exchange** (`str`) – Destination exchange.
- **currency** (`str`) – Underlying currency.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.contract.CFD(symbol="", exchange="", currency="", **kwargs)`

Contract For Difference.

**Parameters**

- **symbol** (*str*) – Symbol name.
- **exchange** (*str*) – Destination exchange.
- **currency** (*str*) – Underlying currency.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.contract.Commodity(symbol="", exchange="", currency="", **kwargs)
```

Commodity.

**Parameters**

- **symbol** (*str*) – Symbol name.
- **exchange** (*str*) – Destination exchange.
- **currency** (*str*) – Underlying currency.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.contract.Bond(**kwargs)
```

Bond.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.



**Return type***object*

```
class ib_insync.contract.FuturesOption(symbol="", lastTradeDateOrContractMonth="", strike=0.0,
                                         right="", exchange="", multiplier="", currency="", **kwargs)
```

Option on a futures contract.

**Parameters**

- **symbol** (*str*) – Symbol name.
- **lastTradeDateOrContractMonth** (*str*) – The option's last trading day or contract month.
  - YYYYMM format: To specify last month
  - YYYYMMDD format: To specify last trading day
- **strike** (*float*) – The option's strike price.
- **right** (*str*) – Put or call option. Valid values are 'P', 'PUT', 'C' or 'CALL'.
- **exchange** (*str*) – Destination exchange.
- **multiplier** (*str*) – The contract multiplier.
- **currency** (*str*) – Underlying currency.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type***dict***nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type***dict***tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type***tuple***update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type***object*

```
class ib_insync.contract.MutualFund(**kwargs)
```

Mutual fund.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type***dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class ib\_insync.contract.Warrant(\*\*kwargs)**

Warrant option.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class ib\_insync.contract.Bag(\*\*kwargs)**

Bag contract.

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

```
class ib_insync.contract.Crypto(symbol="", exchange="", currency="", **kwargs)
```

Crypto currency contract.

**Parameters**

- **symbol** (`str`) – Symbol name.
- **exchange** (`str`) – Destination exchange.
- **currency** (`str`) – Underlying currency.

**dict()**

Return dataclass values as `dict`. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as `dict`.

**Return type**

`dict`

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

```
class ib_insync.contract.TagValue(tag, value)
```

Create new instance of TagValue(tag, value)

**tag:** `str`

**value:** `str`

```
class ib_insync.contract.ComboLeg(conId=0, ratio=0, action="", exchange="", openClose=0,
                                   shortSaleSlot=0, designatedLocation="", exemptCode=-1)
```

**conId:** `int` = 0

```
ratio:  int = 0
action:  str = ''
exchange: str = ''
openClose: int = 0
shortSaleSlot: int = 0
designatedLocation: str = ''
exemptCode: int = -1
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.contract.DeltaNeutralContract(conId=0, delta=0.0, price=0.0)
```

```
conId:  int = 0
delta:  float = 0.0
price:  float = 0.0
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

```
class ib_insync.contract.TradingSession(start, end)
```

Create new instance of TradingSession(start, end)

**start:** `datetime`

**end:** `datetime`

```
class ib_insync.contract.ContractDetails(contract=None, marketName="", minTick=0.0, orderTypes="",
                                         validExchanges="", priceMagnifier=0, underConId=0,
                                         longName="", contractMonth="", industry="", category="",
                                         subcategory="", timeZoneId="", tradingHours="",
                                         liquidHours="", evRule="", evMultiplier=0,
                                         mdSizeMultiplier=1, aggGroup=0, underSymbol="",
                                         underSecType="", marketRuleIds="", secIdList=<factory>,
                                         realExpirationDate="", lastTradeTime="", stockType="",
                                         minSize=0.0, sizeIncrement=0.0, suggestedSizeIncrement=0.0,
                                         cusip="", ratings="", descAppend="", bondType="",
                                         couponType="", callable=False, putable=False, coupon=0,
                                         convertible=False, maturity="", issueDate="",
                                         nextOptionDate="", nextOptionType="",
                                         nextOptionPartial=False, notes="")
```

**contract:** `Optional[Contract]` = None

**marketName:** `str` = ''

**minTick:** `float` = 0.0

**orderTypes:** `str` = ''

**validExchanges:** `str` = ''

**priceMagnifier:** `int` = 0

**underConId:** `int` = 0

**longName:** `str` = ''

**contractMonth:** `str` = ''

**industry:** `str` = ''

**category:** `str` = ''

**subcategory:** `str` = ''

```
timeZoneId: str = ''
tradingHours: str = ''
liquidHours: str = ''
evRule: str = ''
evMultiplier: int = 0
mdSizeMultiplier: int = 1
aggGroup: int = 0
underSymbol: str = ''
underSecType: str = ''
marketRuleIds: str = ''
secIdList: List[TagValue]
realExpirationDate: str = ''
lastTradeTime: str = ''
stockType: str = ''
minSize: float = 0.0
sizeIncrement: float = 0.0
suggestedSizeIncrement: float = 0.0
cusip: str = ''
ratings: str = ''
descAppend: str = ''
bondType: str = ''
couponType: str = ''
callable: bool = False
putable: bool = False
coupon: float = 0
convertible: bool = False
maturity: str = ''
issueDate: str = ''
nextOptionDate: str = ''
nextOptionType: str = ''
nextOptionPartial: bool = False
```

**notes:** `str` = ''

**tradingSessions()**

**Return type**

`List[TradingSession]`

**liquidSessions()**

**Return type**

`List[TradingSession]`

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

`dict`

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

**class** `ib_insync.contract.ContractDescription(contract=None, derivativeSecTypes=<factory>)`

**contract:** `Optional[Contract]` = None

**derivativeSecTypes:** `List[str]`

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

`dict`

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.contract.ScanData`(rank, contractDetails, distance, benchmark, projection, legsStr)

**rank:** *int*

**contractDetails:** *ContractDetails*

**distance:** *str*

**benchmark:** *str*

**projection:** *str*

**legsStr:** *str*

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

## 2.5 Ticker

Access to realtime market information.



```
class ib_insync.ticker.Ticker(contract=None, time=None, marketDataType=1, minTick=nan, bid=nan,
    bidSize=nan, bidExchange="", ask=nan, askSize=nan, askExchange="",
    last=nan, lastSize=nan, lastExchange="", prevBid=nan, prevBidSize=nan,
    prevAsk=nan, prevAskSize=nan, prevLast=nan, prevLastSize=nan,
    volume=nan, open=nan, high=nan, low=nan, close=nan, vwap=nan,
    low13week=nan, high13week=nan, low26week=nan, high26week=nan,
    low52week=nan, high52week=nan, bidYield=nan, askYield=nan,
    lastYield=nan, markPrice=nan, halted=nan, rtHistVolatility=nan,
    rtVolume=nan, rtTradeVolume=nan, rtTime=None, avVolume=nan,
    tradeCount=nan, tradeRate=nan, volumeRate=nan, shortableShares=nan,
    indexFuturePremium=nan, futuresOpenInterest=nan, putOpenInterest=nan,
    callOpenInterest=nan, putVolume=nan, callVolume=nan,
    avOptionVolume=nan, histVolatility=nan, impliedVolatility=nan,
    dividends=None, fundamentalRatios=None, ticks=<factory>,
    tickByTicks=<factory>, domBids=<factory>, domAsks=<factory>,
    domTicks=<factory>, bidGreeks=None, askGreeks=None, lastGreeks=None,
    modelGreeks=None, auctionVolume=nan, auctionPrice=nan,
    auctionImbalance=nan, regulatoryImbalance=nan, bboExchange="",
    snapshotPermissions=0)
```

Current market data such as bid, ask, last price, etc. for a contract.

Streaming level-1 ticks of type [TickData](#) are stored in the ticks list.

Streaming level-2 ticks of type [MktDepthData](#) are stored in the domTicks list. The order book (DOM) is available as lists of [DOMLevel](#) in domBids and domAsks.

Streaming tick-by-tick ticks are stored in tickByTicks.

For options the [OptionComputation](#) values for the bid, ask, resp. last price are stored in the bidGreeks, askGreeks resp. lastGreeks attributes. There is also modelGreeks that conveys the greeks as calculated by Interactive Brokers' option model.

#### Events:

- updateEvent (ticker: [Ticker](#))

```
events: ClassVar = ('updateEvent',)
```

```
contract: Optional[Contract] = None
```

```
time: Optional[datetime] = None
```

```
marketDataType: int = 1
```

```
minTick: float = nan
```

```
bid: float = nan
```

```
bidSize: float = nan
```

```
bidExchange: str = ''
```

```
ask: float = nan
```

```
askSize: float = nan
```

```
askExchange: str = ''
```

```
last: float = nan
```

```
lastSize: float = nan
lastExchange: str = ''
prevBid: float = nan
prevBidSize: float = nan
prevAsk: float = nan
prevAskSize: float = nan
prevLast: float = nan
prevLastSize: float = nan
volume: float = nan
open: float = nan
high: float = nan
low: float = nan
close: float = nan
vwap: float = nan
low13week: float = nan
high13week: float = nan
low26week: float = nan
high26week: float = nan
low52week: float = nan
high52week: float = nan
bidYield: float = nan
askYield: float = nan
lastYield: float = nan
markPrice: float = nan
halted: float = nan
rtHistVolatility: float = nan
rtVolume: float = nan
rtTradeVolume: float = nan
rtTime: Optional[datetime] = None
avVolume: float = nan
tradeCount: float = nan
```

```

tradeRate: float = nan
volumeRate: float = nan
shortableShares: float = nan
indexFuturePremium: float = nan
futuresOpenInterest: float = nan
putOpenInterest: float = nan
callOpenInterest: float = nan
putVolume: float = nan
callVolume: float = nan
avOptionVolume: float = nan
histVolatility: float = nan
impliedVolatility: float = nan
dividends: Optional[Dividends] = None
fundamentalRatios: Optional[FundamentalRatios] = None
ticks: List[TickData]
tickByTicks: List[Union[TickByTickAllLast, TickByTickBidAsk, TickByTickMidPoint]]
domBids: List[DOMLevel]
domAsks: List[DOMLevel]
domTicks: List[MktDepthData]
bidGreeks: Optional[OptionComputation] = None
askGreeks: Optional[OptionComputation] = None
lastGreeks: Optional[OptionComputation] = None
modelGreeks: Optional[OptionComputation] = None
auctionVolume: float = nan
auctionPrice: float = nan
auctionImbalance: float = nan
regulatoryImbalance: float = nan
bboExchange: str = ''
snapshotPermissions: int = 0

```

**hasBidAsk()**

See if this ticker has a valid bid and ask.

**Return type**

*bool*

**midpoint()**

Return average of bid and ask, or NaN if no valid bid and ask are available.

**Return type**

*float*

**marketPrice()**

Return the first available one of :rtype: *float*

- last price if within current bid/ask or no bid/ask available;
- average of bid and ask (midpoint).

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.ticker.TickerUpdateEvent`(*name=""*, *\_with\_error\_done\_events=True*)

**trades()**

Emit trade ticks.

**Return type**

*Tickfilter*

**bids()**

Emit bid ticks.

**Return type**

*Tickfilter*

**asks()**

Emit ask ticks.

**Return type***Tickfilter***bidasks()**

Emit bid and ask ticks.

**Return type***Tickfilter***midpoints()**

Emit midpoint ticks.

**Return type***Tickfilter***class** `ib_insync.ticker.Tickfilter`(*tickTypes*, *source=None*)Tick filtering event operators that emit(*time*, *price*, *size*).**on\_source**(*ticker*)

Emit a new value to all connected listeners.

**Parameters****args** – Argument values to emit to listeners.**timebars**(*timer*)Aggregate ticks into time bars, where the timing of new bars is derived from a timer event. Emits a completed *Bar*.This event stores a *BarList* of all created bars in the *bars* property.**Parameters****timer** (*Event*) – Event for timing when a new bar starts.**Return type***TimeBars***tickbars**(*count*)Aggregate ticks into bars that have the same number of ticks. Emits a completed *Bar*.This event stores a *BarList* of all created bars in the *bars* property.**Parameters****count** (*int*) – Number of ticks to use to form one bar.**Return type***TickBars***volumebars**(*volume*)Aggregate ticks into bars that have the same volume. Emits a completed *Bar*.This event stores a *BarList* of all created bars in the *bars* property.**Parameters****count** – Number of ticks to use to form one bar.**Return type***VolumeBars***class** `ib_insync.ticker.Midpoints`(*tickTypes*, *source=None*)

**on\_source**(*ticker*)

Emit a new value to all connected listeners.

**Parameters**

**args** – Argument values to emit to listeners.

**class** `ib_insync.ticker.Bar`(*time*, *open=nan*, *high=nan*, *low=nan*, *close=nan*, *volume=0*, *count=0*)

**time:** `Optional[datetime]`

**open:** `float = nan`

**high:** `float = nan`

**low:** `float = nan`

**close:** `float = nan`

**volume:** `int = 0`

**count:** `int = 0`

**class** `ib_insync.ticker.BarList`(\**args*)

**class** `ib_insync.ticker.TimeBars`(*timer*, *source=None*)

Aggregate ticks into time bars, where the timing of new bars is derived from a timer event. Emits a completed [Bar](#).

This event stores a [BarList](#) of all created bars in the `bars` property.

**Parameters**

**timer** – Event for timing when a new bar starts.

**bars:** [BarList](#)

**on\_source**(*time*, *price*, *size*)

Emit a new value to all connected listeners.

**Parameters**

**args** – Argument values to emit to listeners.

**class** `ib_insync.ticker.TickBars`(*count*, *source=None*)

Aggregate ticks into bars that have the same number of ticks. Emits a completed [Bar](#).

This event stores a [BarList](#) of all created bars in the `bars` property.

**Parameters**

**count** – Number of ticks to use to form one bar.

**bars:** [BarList](#)

**on\_source**(*time*, *price*, *size*)

Emit a new value to all connected listeners.

**Parameters**

**args** – Argument values to emit to listeners.

**class** `ib_insync.ticker.VolumeBars`(*volume*, *source=None*)

Aggregate ticks into bars that have the same volume. Emits a completed [Bar](#).

This event stores a [BarList](#) of all created bars in the `bars` property.

**Parameters****count** – Number of ticks to use to form one bar.**bars:** *BarList***on\_source**(*time, price, size*)

Emit a new value to all connected listeners.

**Parameters****args** – Argument values to emit to listeners.

## 2.6 Objects

Object hierarchy.

```
class ib_insync.objects.ScannerSubscription(numberOfRows=-1, instrument="", locationCode="",
                                           scanCode="", abovePrice=1.7976931348623157e+308,
                                           belowPrice=1.7976931348623157e+308,
                                           aboveVolume=2147483647,
                                           marketCapAbove=1.7976931348623157e+308,
                                           marketCapBelow=1.7976931348623157e+308,
                                           moodyRatingAbove="", moodyRatingBelow="",
                                           spRatingAbove="", spRatingBelow="",
                                           maturityDateAbove="", maturityDateBelow="",
                                           couponRateAbove=1.7976931348623157e+308,
                                           couponRateBelow=1.7976931348623157e+308,
                                           excludeConvertible=False,
                                           averageOptionVolumeAbove=2147483647,
                                           scannerSettingPairs="", stockTypeFilter="")
```

```
numberOfRows:  int = -1
instrument:     str = ''
locationCode:  str = ''
scanCode:      str = ''
abovePrice:    float = 1.7976931348623157e+308
belowPrice:    float = 1.7976931348623157e+308
aboveVolume:   int = 2147483647
marketCapAbove: float = 1.7976931348623157e+308
marketCapBelow: float = 1.7976931348623157e+308
moodyRatingAbove: str = ''
moodyRatingBelow: str = ''
spRatingAbove:  str = ''
spRatingBelow:  str = ''
```

```
maturityDateAbove: str = ''
maturityDateBelow: str = ''
couponRateAbove: float = 1.7976931348623157e+308
couponRateBelow: float = 1.7976931348623157e+308
excludeConvertible: bool = False
averageOptionVolumeAbove: int = 2147483647
scannerSettingPairs: str = ''
stockTypeFilter: str = ''
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**  
*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**  
*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**  
*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**  
*object*

```
class ib_insync.objects.SoftDollarTier(name="", val="", displayName="")
```

```
name: str = ''
val: str = ''
displayName: str = ''
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**  
*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**  
*dict*



**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

```
class ib_insync.objects.Execution(execId="", time=datetime.datetime(1970, 1, 0, 0,
                                tzinfo=datetime.timezone.utc), acctNumber="", exchange="", side="",
                                shares=0.0, price=0.0, permId=0, clientId=0, orderId=0,
                                liquidation=0, cumQty=0.0, avgPrice=0.0, orderRef="", evRule="",
                                evMultiplier=0.0, modelCode="", lastLiquidity=0,
                                pendingPriceRevision=False)

execId:  str = ''
time:    datetime = datetime.datetime(1970, 1, 0, 0, 0, tzinfo=datetime.timezone.utc)
acctNumber:  str = ''
exchange:    str = ''
side:        str = ''
shares:      float = 0.0
price:       float = 0.0
permId:      int = 0
clientId:    int = 0
orderId:     int = 0
liquidation: int = 0
cumQty:      float = 0.0
avgPrice:    float = 0.0
orderRef:    str = ''
evRule:      str = ''
evMultiplier: float = 0.0
modelCode:   str = ''
lastLiquidity: int = 0
pendingPriceRevision: bool = False
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.CommissionReport(execId="", commission=0.0, currency="", realizedPNL=0.0,
                                         yield_=0.0, yieldRedemptionDate=0)
```

```
execId: str = ''
```

```
commission: float = 0.0
```

```
currency: str = ''
```

```
realizedPNL: float = 0.0
```

```
yield_: float = 0.0
```

```
yieldRedemptionDate: int = 0
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.ExecutionFilter(clientId=0, acctCode="", time="", symbol="", secType="",
                                       exchange="", side="")
```

**clientId:** *int* = 0

**acctCode:** *str* = ''

**time:** *str* = ''

**symbol:** *str* = ''

**secType:** *str* = ''

**exchange:** *str* = ''

**side:** *str* = ''

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.BarData(date=datetime.datetime(1970, 1, 1, 0, 0, tzinfo=datetime.timezone.utc),
                                open=0.0, high=0.0, low=0.0, close=0.0, volume=0, average=0.0,
                                barCount=0)
```

**date:** *Union[date, datetime]* = `datetime.datetime(1970, 1, 1, 0, 0, tzinfo=datetime.timezone.utc)`

**open:** *float* = 0.0

**high:** *float* = 0.0

**low:** `float` = 0.0

**close:** `float` = 0.0

**volume:** `float` = 0

**average:** `float` = 0.0

**barCount:** `int` = 0

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

`dict`

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

```
class ib_insync.objects.RealTimeBar(time=datetime.datetime(1970, 1, 1, 0, 0,
tzinfo=datetime.timezone.utc), endTime=-1, open_=0.0, high=0.0,
low=0.0, close=0.0, volume=0.0, wap=0.0, count=0)
```

**time:** `datetime` = `datetime.datetime(1970, 1, 1, 0, 0, tzinfo=datetime.timezone.utc)`

**endTime:** `int` = -1

**open\_:** `float` = 0.0

**high:** `float` = 0.0

**low:** `float` = 0.0

**close:** `float` = 0.0

**volume:** `float` = 0.0

**wap:** `float` = 0.0

**count:** `int` = 0

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.objects.TickAttrib`(*canAutoExecute=False, pastLimit=False, preOpen=False*)

**canAutoExecute:** `bool` = `False`

**pastLimit:** `bool` = `False`

**preOpen:** `bool` = `False`

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

**class** `ib_insync.objects.TickAttribBidAsk`(*bidPastLow=False, askPastHigh=False*)

**bidPastLow:** `bool` = `False`

**askPastHigh:** `bool` = `False`

**dict()**

Return dataclass values as `dict`. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as `dict`.

**Return type**

`dict`

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

**class** `ib_insync.objects.TickAttribLast` (*pastLimit=False, unreported=False*)

**pastLimit:** `bool` = `False`

**unreported:** `bool` = `False`

**dict()**

Return dataclass values as `dict`. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as `dict`.

**Return type**

`dict`

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

---

```
class ib_insync.objects.HistogramData(price=0.0, count=0)
```

```
    price: float = 0.0
```

```
    count: int = 0
```

```
    dict()
```

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

```
    nonDefaults()
```

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

```
    tuple()
```

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

```
    update(*srcObjs, **kwargs)
```

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.NewsProvider(code="", name="")
```

```
    code: str = ''
```

```
    name: str = ''
```

```
    dict()
```

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

```
    nonDefaults()
```

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

```
    tuple()
```

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

```
    update(*srcObjs, **kwargs)
```

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.DepthMktDataDescription(exchange="", secType="", listingExch="",
                                                  serviceDataType="", aggGroup=2147483647)
```

```
    exchange: str = ''
```

```
    secType: str = ''
```

```
    listingExch: str = ''
```

```
    serviceDataType: str = ''
```

```
    aggGroup: int = 2147483647
```

```
    dict()
```

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

```
    nonDefaults()
```

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

```
    tuple()
```

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

```
    update(*srcObjs, **kwargs)
```

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.PnL(account="", modelCode="", dailyPnL=nan, unrealizedPnL=nan,
                             realizedPnL=nan)
```

```
    account: str = ''
```

```
    modelCode: str = ''
```

```
    dailyPnL: float = nan
```

```
    unrealizedPnL: float = nan
```

```
    realizedPnL: float = nan
```

```
    dict()
```

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*



**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.TradeLogEntry(time, status="", message="", errorCode=0)
```

```
time:  datetime
```

```
status: str = ''
```

```
message: str = ''
```

```
errorCode: int = 0
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.PnLSingle(account="", modelCode="", conId=0, dailyPnL=nan,  
                                unrealizedPnL=nan, realizedPnL=nan, position=0, value=nan)
```

```
account:  str = ''
```

```
modelCode: str = ''
```

```
conId: int = 0
dailyPnL: float = nan
unrealizedPnL: float = nan
realizedPnL: float = nan
position: int = 0
value: float = nan
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**  
*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**  
*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**  
*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**  
*object*

```
class ib_insync.objects.HistoricalSession(startDateTime="", endDateTime="", refDate="")
```

```
    startDateTime: str = ''
```

```
    endDateTime: str = ''
```

```
    refDate: str = ''
```

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**  
*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**  
*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**  
*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.HistoricalSchedule(startDateTime="", endDateTime="", timeZone="",
                                           sessions=<factory>)
```

**startDateTime: str = ''**

**endDateTime: str = ''**

**timeZone: str = ''**

**sessions: List[HistoricalSession]**

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*

```
class ib_insync.objects.WshEventData(conId=2147483647, filter="", fillWatchlist=False,
                                     fillPortfolio=False, fillCompetitors=False, startDate="", endDate="",
                                     totalLimit=2147483647)
```

**conId: int = 2147483647**

**filter: str = ''**

**fillWatchlist: bool = False**

**fillPortfolio: bool = False**

**fillCompetitors: bool = False**

**startDate: str = ''**

**endDate: str = ''**

**totalLimit:** `int` = 2147483647

**dict()**

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

`dict`

**tuple()**

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

**class** `ib_insync.objects.AccountValue`(*account, tag, value, currency, modelCode*)

Create new instance of AccountValue(account, tag, value, currency, modelCode)

**account:** `str`

**tag:** `str`

**value:** `str`

**currency:** `str`

**modelCode:** `str`

**class** `ib_insync.objects.TickData`(*time, tickType, price, size*)

Create new instance of TickData(time, tickType, price, size)

**time:** `datetime`

**tickType:** `int`

**price:** `float`

**size:** `float`

**class** `ib_insync.objects.HistoricalTick`(*time, price, size*)

Create new instance of HistoricalTick(time, price, size)

**time:** `datetime`

**price:** `float`

**size:** `float`

```
class ib_insync.objects.HistoricalTickBidAsk(time, tickAttribBidAsk, priceBid, priceAsk, sizeBid,
                                             sizeAsk)
```

Create new instance of HistoricalTickBidAsk(*time, tickAttribBidAsk, priceBid, priceAsk, sizeBid, sizeAsk*)

```
time: datetime
tickAttribBidAsk: TickAttribBidAsk
priceBid: float
priceAsk: float
sizeBid: float
sizeAsk: float
```

```
class ib_insync.objects.HistoricalTickLast(time, tickAttribLast, price, size, exchange,
                                           specialConditions)
```

Create new instance of HistoricalTickLast(*time, tickAttribLast, price, size, exchange, specialConditions*)

```
time: datetime
tickAttribLast: TickAttribLast
price: float
size: float
exchange: str
specialConditions: str
```

```
class ib_insync.objects.TickByTickAllLast(tickType, time, price, size, tickAttribLast, exchange,
                                           specialConditions)
```

Create new instance of TickByTickAllLast(*tickType, time, price, size, tickAttribLast, exchange, specialConditions*)

```
tickType: int
time: datetime
price: float
size: float
tickAttribLast: TickAttribLast
exchange: str
specialConditions: str
```

```
class ib_insync.objects.TickByTickBidAsk(time, bidPrice, askPrice, bidSize, askSize, tickAttribBidAsk)
```

Create new instance of TickByTickBidAsk(*time, bidPrice, askPrice, bidSize, askSize, tickAttribBidAsk*)

```
time: datetime
bidPrice: float
askPrice: float
```

**bidSize:** `float`

**askSize:** `float`

**tickAttribBidAsk:** `TickAttribBidAsk`

**class** `ib_insync.objects.TickByTickMidPoint`(*time, midPoint*)

Create new instance of `TickByTickMidPoint`(*time, midPoint*)

**time:** `datetime`

**midPoint:** `float`

**class** `ib_insync.objects.MktDepthData`(*time, position, marketMaker, operation, side, price, size*)

Create new instance of `MktDepthData`(*time, position, marketMaker, operation, side, price, size*)

**time:** `datetime`

**position:** `int`

**marketMaker:** `str`

**operation:** `int`

**side:** `int`

**price:** `float`

**size:** `float`

**class** `ib_insync.objects.DOMLevel`(*price, size, marketMaker*)

Create new instance of `DOMLevel`(*price, size, marketMaker*)

**price:** `float`

**size:** `float`

**marketMaker:** `str`

**class** `ib_insync.objects.PriceIncrement`(*lowEdge, increment*)

Create new instance of `PriceIncrement`(*lowEdge, increment*)

**lowEdge:** `float`

**increment:** `float`

**class** `ib_insync.objects.PortfolioItem`(*contract, position, marketPrice, marketValue, averageCost, unrealizedPNL, realizedPNL, account*)

Create new instance of `PortfolioItem`(*contract, position, marketPrice, marketValue, averageCost, unrealizedPNL, realizedPNL, account*)

**contract:** `Contract`

**position:** `float`

**marketPrice:** `float`

**marketValue:** `float`

**averageCost:** `float`

```

    unrealizedPNL: float

    realizedPNL: float

    account: str

class ib_insync.objects.Position(account, contract, position, avgCost)
    Create new instance of Position(account, contract, position, avgCost)

    account: str

    contract: Contract

    position: float

    avgCost: float

class ib_insync.objects.Fill(contract, execution, commissionReport, time)
    Create new instance of Fill(contract, execution, commissionReport, time)

    contract: Contract

    execution: Execution

    commissionReport: CommissionReport

    time: datetime

class ib_insync.objects.OptionComputation(tickAttrib, impliedVol, delta, optPrice, pvDividend, gamma,
                                           vega, theta, undPrice)
    Create new instance of OptionComputation(tickAttrib, impliedVol, delta, optPrice, pvDividend, gamma, vega,
    theta, undPrice)

    tickAttrib: int

    impliedVol: Optional[float]

    delta: Optional[float]

    optPrice: Optional[float]

    pvDividend: Optional[float]

    gamma: Optional[float]

    vega: Optional[float]

    theta: Optional[float]

    undPrice: Optional[float]

class ib_insync.objects.OptionChain(exchange, underlyingConId, tradingClass, multiplier, expirations,
                                    strikes)
    Create new instance of OptionChain(exchange, underlyingConId, tradingClass, multiplier, expirations, strikes)

    exchange: str

    underlyingConId: int

    tradingClass: str

```

```
    multiplier: str
    expirations: List[str]
    strikes: List[float]

class ib_insync.objects.Dividends(past12Months, next12Months, nextDate, nextAmount)
    Create new instance of Dividends(past12Months, next12Months, nextDate, nextAmount)
    past12Months: Optional[float]
    next12Months: Optional[float]
    nextDate: Optional[date]
    nextAmount: Optional[float]

class ib_insync.objects.NewsArticle(articleType, articleText)
    Create new instance of NewsArticle(articleType, articleText)
    articleType: int
    articleText: str

class ib_insync.objects.HistoricalNews(time, providerCode, articleId, headline)
    Create new instance of HistoricalNews(time, providerCode, articleId, headline)
    time: datetime
    providerCode: str
    articleId: str
    headline: str

class ib_insync.objects.NewsTick(timeStamp, providerCode, articleId, headline, extraData)
    Create new instance of NewsTick(timeStamp, providerCode, articleId, headline, extraData)
    timeStamp: int
    providerCode: str
    articleId: str
    headline: str
    extraData: str

class ib_insync.objects.NewsBulletin(msgId, msgType, message, origExchange)
    Create new instance of NewsBulletin(msgId, msgType, message, origExchange)
    msgId: int
    msgType: int
    message: str
    origExchange: str
```



```

class ib_insync.objects.FamilyCode(accountID, familyCodeStr)
    Create new instance of FamilyCode(accountID, familyCodeStr)
    accountID: str
    familyCodeStr: str

class ib_insync.objects.SmartComponent(bitNumber, exchange, exchangeLetter)
    Create new instance of SmartComponent(bitNumber, exchange, exchangeLetter)
    bitNumber: int
    exchange: str
    exchangeLetter: str

class ib_insync.objects.ConnectionStats(startTime, duration, numBytesRecv, numBytesSent,
                                         numMsgRecv, numMsgSent)
    Create new instance of ConnectionStats(startTime, duration, numBytesRecv, numBytesSent, numMsgRecv,
    numMsgSent)
    startTime: float
    duration: float
    numBytesRecv: int
    numBytesSent: int
    numMsgRecv: int
    numMsgSent: int

class ib_insync.objects.BarDataList(*args)
    List of BarData that also stores all request parameters.
    Events:
        • updateEvent (bars: BarDataList, hasNewBar: bool)
    reqId: int
    contract: Contract
    endDateTime: Union[datetime, date, str, None]
    durationStr: str
    barSizeSetting: str
    whatToShow: str
    useRTH: bool
    formatDate: int
    keepUpToDate: bool
    chartOptions: List[TagValue]

```

```
class ib_insync.objects.RealTimeBarList(*args)
```

List of *RealTimeBar* that also stores all request parameters.

Events:

- `updateEvent` (bars: *RealTimeBarList*, hasNewBar: bool)

`reqId`: `int`

`contract`: *Contract*

`barSize`: `int`

`whatToShow`: `str`

`useRTH`: `bool`

`realTimeBarsOptions`: `List[TagValue]`

```
class ib_insync.objects.ScanDataList(*args)
```

List of *ScanData* that also stores all request parameters.

Events:

- `updateEvent` (*ScanDataList*)

`reqId`: `int`

`subscription`: *ScannerSubscription*

`scannerSubscriptionOptions`: `List[TagValue]`

`scannerSubscriptionFilterOptions`: `List[TagValue]`

```
class ib_insync.objects.DynamicObject(**kwargs)
```

```
class ib_insync.objects.FundamentalRatios(**kwargs)
```

See: [https://web.archive.org/web/20200725010343/https://interactivebrokers.github.io/tws-api/fundamental\\_ratios\\_tags.html](https://web.archive.org/web/20200725010343/https://interactivebrokers.github.io/tws-api/fundamental_ratios_tags.html)

```
class ib_insync.wrapper.RequestError(reqId, code, message)
```

Exception to raise when the API reports an error that can be tied to a single request.

Parameters

- `reqId` (`int`) – Original request ID.
- `code` (`int`) – Original error code.
- `message` (`str`) – Original error message.

## 2.7 Utilities

Utilities.

```
ib_insync.util.globalErrorEvent(*args) = Event<Event, []>
```

Event to emit global exceptions.

`ib_insync.util.df(objs, labels=None)`

Create pandas DataFrame from the sequence of same-type objects.

**Parameters**

**labels** (`Optional[List[str]]`) – If supplied, retain only the given labels and drop the rest.

`ib_insync.util.dataclassAsDict(obj)`

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

`dict`

`ib_insync.util.dataclassAsTuple(obj)`

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

`tuple`

`ib_insync.util.dataclassNonDefaults(obj)`

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

`dict`

`ib_insync.util.dataclassUpdate(obj, *srcObjs, **kwargs)`

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

`ib_insync.util.dataclassRepr(obj)`

Provide a culled representation of the given dataclass instance, showing only the fields with a non-default value.

**Return type**

`str`

`ib_insync.util.isnamedtupleinstance(x)`

From <https://stackoverflow.com/a/2166841/6067848>

`ib_insync.util.tree(obj)`

Convert object to a tree of lists, dicts and simple values. The result can be serialized to JSON.

`ib_insync.util.barplot(bars, title="", upColor='blue', downColor='red')`

Create candlestick plot for the given bars. The bars can be given as a DataFrame or as a list of bar objects.

`ib_insync.util.allowCtrlC()`

Allow Control-C to end program.

`ib_insync.util.logToFile(path, level=20)`

Create a log handler that logs to the given file.

`ib_insync.util.logToConsole(level=20)`

Create a log handler that logs to the console.

`ib_insync.util.isNaN(x)`

Not a number test.

**Return type**

`bool`

`ib_insync.util.formatSI(n)`

Format the integer or float `n` to 3 significant digits + SI prefix.

**Return type**

`str`

`class ib_insync.util.timeit(title='Run')`

Context manager for timing.

`ib_insync.util.run(*awaitables, timeout=None)`

By default run the event loop forever.

When awaitables (like Tasks, Futures or coroutines) are given then run the event loop until each has completed and return their results.

An optional timeout (in seconds) can be given that will raise `asyncio.TimeoutError` if the awaitables are not ready within the timeout period.

`ib_insync.util.schedule(time, callback, *args)`

Schedule the callback to be run at the given time with the given arguments. This will return the Event Handle.

**Parameters**

- **time** (`Union[time, datetime]`) – Time to run callback. If given as `datetime.time` then use today as date.
- **callback** (`Callable`) – Callable scheduled to run.
- **args** – Arguments for to call callback with.

`ib_insync.util.sleep(secs=0.02)`

Wait for the given amount of seconds while everything still keeps processing in the background. Never use `time.sleep()`.

**Parameters**

**secs** (`float`) – Time in seconds to wait.

**Return type**

`bool`

`ib_insync.util.timeRange(start, end, step)`

Iterator that waits periodically until certain time points are reached while yielding those time points.

**Parameters**

- **start** (`Union[time, datetime]`) – Start time, can be specified as `datetime.datetime`, or as `datetime.time` in which case today is used as the date
- **end** (`Union[time, datetime]`) – End time, can be specified as `datetime.datetime`, or as `datetime.time` in which case today is used as the date
- **step** (`float`) – The number of seconds of each period

**Return type**

`Iterator[datetime]`

`ib_insync.util.waitUntil(t)`

Wait until the given time `t` is reached.

**Parameters**

**t** (`Union[time, datetime]`) – The time `t` can be specified as `datetime.datetime`, or as `datetime.time` in which case today is used as the date.

**Return type**`bool`**async** `ib_insync.util.timeRangeAsync(start, end, step)`Async version of `timeRange()`.**Return type**`AsyncIterator[datetime]`**async** `ib_insync.util.waitUntilAsync(t)`Async version of `waitUntil()`.**Return type**`bool``ib_insync.util.patchAsyncio()`

Patch asyncio to allow nested event loops.

`ib_insync.util.getLoop()`

Get the asyncio event loop for the current thread.

`ib_insync.util.startLoop()`

Use nested asyncio event loop for Jupyter notebooks.

`ib_insync.util.useQt(qtLib='PyQt5', period=0.01)`

Run combined Qt5/asyncio event loop.

**Parameters**

- **qtLib** (`str`) – Name of Qt library to use:
  - PyQt5
  - PyQt6
  - PySide2
  - PySide6
- **period** (`float`) – Period in seconds to poll Qt.

`ib_insync.util.formatIBDatetime(t)`

Format date or datetime to string that IB uses.

**Return type**`str``ib_insync.util.parseIBDatetime(s)`

Parse string in IB date or datetime format to datetime.

**Return type**`Union[date, datetime]`

## 2.8 FlexReport

Access to account statement webservice.

**exception** `ib_insync.flexreport.FlexError`

**class** `ib_insync.flexreport.FlexReport` (*token=None, queryId=None, path=None*)

To obtain a token:

- Login to web portal
- Go to Settings
- Click on “Configure Flex Web Service”
- Generate token

Download a report by giving a valid `token` and `queryId`, or load from file by giving a valid `path`.

**data:** `bytes`

**root:** `Element`

**topics()**

Get the set of topics that can be extracted from this report.

**extract** (*topic, parseNumbers=True*)

Extract items of given topic and return as list of objects.

The topic is a string like TradeConfirm, ChangeInDividendAccrual, Order, etc.

**Return type**

`list`

**df** (*topic, parseNumbers=True*)

Same as `extract` but return the result as a pandas DataFrame.

**download** (*token, queryId*)

Download report for the given `token` and `queryId`.

**load** (*path*)

Load report from XML file.

**save** (*path*)

Save report to XML file.

## 2.9 IBC

**class** `ib_insync.ibcontroller.IBC` (*twsversion=0, gateway=False, tradingMode="", twspath="", twsSettingsPath="", ibcPath="", ibcIni="", javaPath="", userid="", password="", fixuserid="", fixpassword="", on2fatimeout=""*)

Programmatic control over starting and stopping TWS/Gateway using IBC (<https://github.com/IbcAlpha/IBC>).

**Parameters**

- **twsversion** (*int*) – (required) The major version number for TWS or gateway.
- **gateway** (*bool*) –

- True = gateway
- False = TWS
- **tradingMode** (*str*) – ‘live’ or ‘paper’.
- **userid** (*str*) – IB account username. It is recommended to set the real username/password in a secured IBC config file.
- **password** (*str*) – IB account password.
- **twsPath** (*str*) – Path to the TWS installation folder. Defaults:
  - Linux: ~/Jts
  - OS X: ~/Applications
  - Windows: C:\Jts
- **twsSettingsPath** (*str*) – Path to the TWS settings folder. Defaults:
  - Linux: ~/Jts
  - OS X: ~/Jts
  - Windows: Not available
- **ibcPath** (*str*) – Path to the IBC installation folder. Defaults:
  - Linux: /opt/ibc
  - OS X: /opt/ibc
  - Windows: C:\IBC
- **ibcIni** (*str*) – Path to the IBC configuration file. Defaults:
  - Linux: ~/ibc/config.ini
  - OS X: ~/ibc/config.ini
  - Windows: %HOMEPATH%\Documents\IBC\config.ini
- **javaPath** (*str*) – Path to Java executable. Default is to use the Java VM included with TWS/gateway.
- **fixuserid** (*str*) – FIX account user id (gateway only).
- **fixpassword** (*str*) – FIX account password (gateway only).
- **on2fatimeout** (*str*) – What to do if 2-factor authentication times out; Can be ‘restart’ or ‘exit’.

This is not intended to be run in a notebook.

To use IBC on Windows, the proactor (or quamash) event loop must have been set:

```
import asyncio
asyncio.set_event_loop(asyncio.ProactorEventLoop())
```

Example usage:

```
ibc = IBC(976, gateway=True, tradingMode='live',
         userid='edemo', password='demouser')
ibc.start()
IB.run()
```

`IbcLogLevel: ClassVar = 10`

`twsversion: int = 0`

`gateway: bool = False`

`tradingMode: str = ''`

`twspath: str = ''`

`twsettingsPath: str = ''`

`ibcPath: str = ''`

`ibcIni: str = ''`

`javaPath: str = ''`

`userid: str = ''`

`password: str = ''`

`fixuserid: str = ''`

`fixpassword: str = ''`

`on2fatimeout: str = ''`

`start()`

Launch TWS/IBG.

`terminate()`

Terminate TWS/IBG.

`async startAsync()`

`async terminateAsync()`

`async monitorAsync()`

`dict()`

Return dataclass values as dict. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

`nonDefaults()`

For a dataclass instance get the fields that are different from the default values and return as dict.

**Return type**

*dict*

`tuple()`

Return dataclass values as tuple. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*



**update**(\*srcObjs, \*\*kwargs)

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

`object`

## 2.10 Watchdog

```
class ib_insync.ibcontroller.Watchdog(controller, ib, host='127.0.0.1', port=7497, clientId=1,
                                     connectTimeout=2, appStartupTime=30, appTimeout=20,
                                     retryDelay=2, readonly=False, account='',
                                     raiseSyncErrors=False, probeContract=Forex('EURUSD',
                                     exchange='IDEALPRO'), probeTimeout=4)
```

Start, connect and watch over the TWS or gateway app and try to keep it up and running. It is intended to be used in an event-driven application that properly initializes itself upon (re-)connect.

It is not intended to be used in a notebook or in imperative-style code. Do not expect Watchdog to magically shield you from reality. Do not use Watchdog unless you understand what it does and doesn't do.

### Parameters

- **controller** (`IBC`) – (required) IBC instance.
- **ib** (`IB`) – (required) IB instance to be used. Do not connect this instance as Watchdog takes care of that.
- **host** (`str`) – Used for connecting IB instance.
- **port** (`int`) – Used for connecting IB instance.
- **clientId** (`int`) – Used for connecting IB instance.
- **connectTimeout** (`float`) – Used for connecting IB instance.
- **readonly** (`bool`) – Used for connecting IB instance.
- **appStartupTime** (`float`) – Time (in seconds) that the app is given to start up. Make sure that it is given ample time.
- **appTimeout** (`float`) – Timeout (in seconds) for network traffic idle time.
- **retryDelay** (`float`) – Time (in seconds) to restart app after a previous failure.
- **probeContract** (`Contract`) – Contract to use for historical data probe requests (default is EURUSD).
- **probeTimeout** (`float`); *Timeout (in seconds)* –

The idea is to wait until there is no traffic coming from the app for a certain amount of time (the `appTimeout` parameter). This triggers a historical request to be placed just to see if the app is still alive and well. If yes, then continue, if no then restart the whole app and reconnect. Restarting will also occur directly on errors 1100 and 100.

Example usage:

```
def onConnected():
    print(ib.accountValues())
```

(continues on next page)

(continued from previous page)

```
ibc = IBC(974, gateway=True, tradingMode='paper')
ib = IB()
ib.connectedEvent += onConnected
watchdog = Watchdog(ibc, ib, port=4002)
watchdog.start()
ib.run()
```

**Events:**

- startingEvent (watchdog: *Watchdog*)
- startedEvent (watchdog: *Watchdog*)
- stoppingEvent (watchdog: *Watchdog*)
- stoppedEvent (watchdog: *Watchdog*)
- softTimeoutEvent (watchdog: *Watchdog*)
- hardTimeoutEvent (watchdog: *Watchdog*)

```
events = ['startingEvent', 'startedEvent', 'stoppingEvent', 'stoppedEvent',
'softTimeoutEvent', 'hardTimeoutEvent']
```

```
controller: IBC
```

```
ib: IB
```

```
host: str = '127.0.0.1'
```

```
port: int = 7497
```

```
clientId: int = 1
```

```
connectTimeout: float = 2
```

```
appStartupTime: float = 30
```

```
appTimeout: float = 20
```

```
retryDelay: float = 2
```

```
readonly: bool = False
```

```
account: str = ''
```

```
raiseSyncErrors: bool = False
```

```
probeContract: Contract = Forex('EURUSD', exchange='IDEALPRO')
```

```
probeTimeout: float = 4
```

```
start()
```

```
stop()
```

```
async runAsync()
```

**dict()**

Return dataclass values as `dict`. This is a non-recursive variant of `dataclasses.asdict`.

**Return type**

*dict*

**nonDefaults()**

For a dataclass instance get the fields that are different from the default values and return as `dict`.

**Return type**

*dict*

**tuple()**

Return dataclass values as `tuple`. This is a non-recursive variant of `dataclasses.astuple`.

**Return type**

*tuple*

**update(\*srcObjs, \*\*kwargs)**

Update fields of the given dataclass object from zero or more dataclass source objects and/or from keyword arguments.

**Return type**

*object*



## NOTEBOOKS

IB-insync can be used in a fully interactive, exploratory way with live data from within a [Jupyter](#) notebook. Here are some recipe notebooks:



## CODE RECIPES

Collection of useful patterns, snippets and recipes.

When using the recipes in a notebook, don't forget to use `util.startLoop()`.

### 4.1 Fetching consecutive historical data

Suppose we want to get the 1 min bar data of Tesla since the very beginning up until now. The best way is to start with now and keep requesting further and further back in time until there is no more data returned.

```
import datetime
from ib_insync import *

ib = IB()
ib.connect('127.0.0.1', 7497, clientId=1)

contract = Stock('TSLA', 'SMART', 'USD')

dt = ''
barsList = []
while True:
    bars = ib.reqHistoricalData(
        contract,
        endDateTime=dt,
        durationStr='10 D',
        barSizeSetting='1 min',
        whatToShow='MIDPOINT',
        useRTH=True,
        formatDate=1)
    if not bars:
        break
    barsList.append(bars)
    dt = bars[0].date
    print(dt)

# save to CSV file
allBars = [b for bars in reversed(barsList) for b in bars]
df = util.df(allBars)
df.to_csv(contract.symbol + '.csv', index=False)
```

## 4.2 Scanner data (blocking)

```
allParams = ib.reqScannerParameters()
print(allParams)

sub = ScannerSubscription(
    instrument='FUT.US',
    locationCode='FUT.GLOBEX',
    scanCode='TOP_PERC_GAIN')
scanData = ib.reqScannerData(sub)
print(scanData)
```

## 4.3 Scanner data (streaming)

```
def onScanData(scanData):
    print(scanData[0])
    print(len(scanData))

sub = ScannerSubscription(
    instrument='FUT.US',
    locationCode='FUT.GLOBEX',
    scanCode='TOP_PERC_GAIN')
scanData = ib.reqScannerSubscription(sub)
scanData.updateEvent += onScanData
ib.sleep(60)
ib.cancelScannerSubscription(scanData)
```

## 4.4 Option calculations

```
option = Option('EOE', '20171215', 490, 'P', 'FTA', multiplier=100)

calc = ib.calculateImpliedVolatility(
    option, optionPrice=6.1, underPrice=525)
print(calc)

calc = ib.calculateOptionPrice(
    option, volatility=0.14, underPrice=525)
print(calc)
```



## 4.5 Order book

```
eurusd = Forex('EURUSD')
ticker = ib.reqMktDepth(eurusd)
while ib.sleep(5):
    print(
        [d.price for d in ticker.domBids],
        [d.price for d in ticker.domAsks])
```

## 4.6 Minimum price increments

```
usdjpy = Forex('USDJPY')
cd = ib.reqContractDetails(usdjpy)[0]
print(cd.marketRuleIds)

rules = [
    ib.reqMarketRule(ruleId)
    for ruleId in cd.marketRuleIds.split(',')
]
print(rules)
```

## 4.7 News articles

```
newsProviders = ib.reqNewsProviders()
print(newsProviders)
codes = '+'.join(np.code for np in newsProviders)

amd = Stock('AMD', 'SMART', 'USD')
ib.qualifyContracts(amd)
headlines = ib.reqHistoricalNews(amd.conId, codes, '', '', 10)
latest = headlines[0]
print(latest)
article = ib.reqNewsArticle(latest.providerCode, latest.articleId)
print(article)
```

## 4.8 News bulletins

```
ib.reqNewsBulletins(True)
ib.sleep(5)
print(ib.newsBulletins())
```

## 4.9 WSH Event Calendar

A Wall Street Horizon subscription is needed to get corporate event data.

```
from ib_insync import *

ib = IB()
ib.connect('127.0.0.1', 7497, clientId=1)

# Get the conId of an instrument (IBM in this case):
ibm = Stock('IBM', 'SMART', 'USD')
ib.qualifyContracts(ibm)
print(ibm.conId) # is 8314

# Get the list of available filters and event types:
meta = ib.getWshMetaData()
print(meta)

# For IBM (with conId=8314) query the:
# - Earnings Dates (wshe_ed)
# - Board of Directors meetings (wshe_bod)
data = WshEventData(
    filter = '''{
        "country": "All",
        "watchlist": ["8314"],
        "limit_region": 10,
        "limit": 10,
        "wshe_ed": "true",
        "wshe_bod": "true"
    }''')
events = ib.getWshEventData(data)
print(events)
```

## 4.10 Dividends

```
contract = Stock('INTC', 'SMART', 'USD')
ticker = ib.reqMktData(contract, '456')
ib.sleep(2)
print(ticker.dividends)
```

Output:

```
Dividends(past12Months=1.2, next12Months=1.2, nextDate=datetime.date(2019, 2, 6),
↪nextAmount=0.3)
```

## 4.11 Fundamental ratios

```
contract = Stock('IBM', 'SMART', 'USD')
ticker = ib.reqMktData(contract, '258')
ib.sleep(2)
print(ticker.fundamentalRatios)
```

## 4.12 Short-lived connections

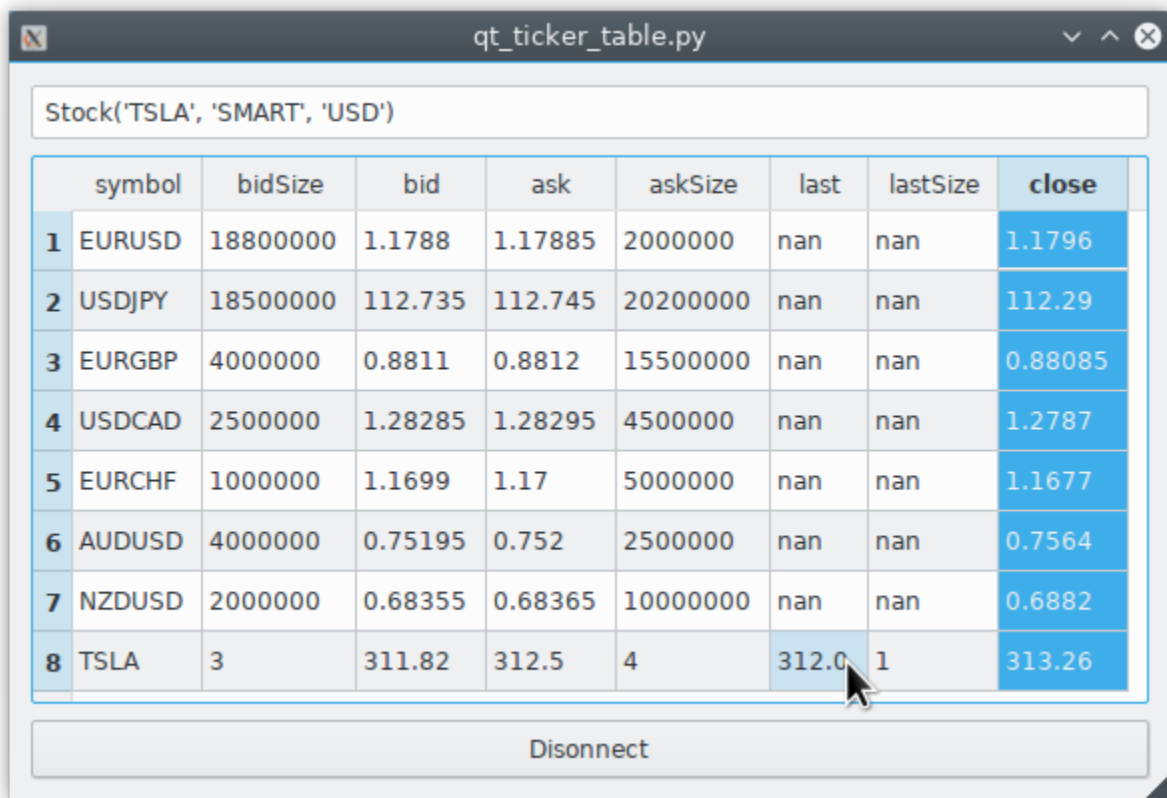
This IB socket protocol is designed to be used for a long-lived connection, lasting a day or so. For short connections, where for example just a few orders are fired off, it is best to add one second of delay before closing the connection. This gives the connection some time to flush the data that has not been sent yet.

```
ib = IB()
ib.connect()

... # create and submit some orders

ib.sleep(1) # added delay
ib.disconnect()
```

## 4.13 Integration with PyQt5 or PySide2



Stock('TSLA', 'SMART', 'USD')

	symbol	bidSize	bid	ask	askSize	last	lastSize	close
1	EURUSD	18800000	1.1788	1.17885	2000000	nan	nan	1.1796
2	USDJPY	18500000	112.735	112.745	20200000	nan	nan	112.29
3	EURGBP	4000000	0.8811	0.8812	15500000	nan	nan	0.88085
4	USDCAD	2500000	1.28285	1.28295	4500000	nan	nan	1.2787
5	EURCHF	1000000	1.1699	1.17	5000000	nan	nan	1.1677
6	AUDUSD	4000000	0.75195	0.752	2500000	nan	nan	0.7564
7	NZDUSD	2000000	0.68355	0.68365	10000000	nan	nan	0.6882
8	TSLA	3	311.82	312.5	4	312.0	1	313.26

Disconnect

This [example](#) of a ticker table shows how to integrate both realtime streaming and synchronous API requests in a single-threaded Qt application. The API requests in this example are `connect` and `ib.qualifyContracts()`; The latter is used to get the `conId` of a contract and use that as a unique key.

The Qt interface will not freeze when a request is ongoing and it is even possible to have multiple outstanding requests at the same time.

This example depends on PyQt5:

```
pip3 install -U PyQt5.
```

It's also possible to use PySide2 instead; To do so uncomment the `PySide2` import and `util.useQt` lines in the example and comment out their `PyQt5` counterparts.

## 4.14 Integration with Tkinter

To integrate with the Tkinter event loop, take a look at [this example app](#).

## 4.15 Integration with PyGame

By calling `ib.sleep` from within the PyGame run loop, `ib_insync` can periodically run for short whiles and keep up to date:

```
import ib_insync as ibi
import pygame

def onTicker(ticker):
    screen.fill(bg_color)
    text = f'bid: {ticker.bid}    ask: {ticker.ask}'
    quote = font.render(text, True, fg_color)
    screen.blit(quote, (40, 40))
    pygame.display.flip()

pygame.init()
screen = pygame.display.set_mode((800, 600))
font = pygame.font.SysFont('arial', 48)
bg_color = (255, 255, 255)
fg_color = (0, 0, 0)

ib = ibi.IB()
ib.connect()
contract = ibi.Forex('EURUSD')
ticker = ib.reqMktData(contract)
ticker.updateEvent += onTicker

running = True
while running:
    # This updates IB-insync:
    ib.sleep(0.03)

    # This updates PyGame:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
            pygame.quit()
```



**SOURCE CODE**





## CHANGELOG

### 6.1 0.9

#### 6.1.1 Version 0.9.86

- Fixed: [issue 588](#): Fixed account summary tag.
- Fixed: [issue 589](#): Fixed more account summary tags.
- pull:598: Year updates

#### 6.1.2 Version 0.9.85

- Fixed: [issue 586](#): Revert socket protocol back to version 176.

#### 6.1.3 Version 0.9.84

- Potential fix for reqWshEventData.

#### 6.1.4 Version 0.9.83

- Added support for WSH (Wall Street Horizon) requests plus the (blocking) convenience methods `getWshMetaData` and `getWshEventData`.
- Updated socket protocol to version 177.
- Added support for Event security type.

#### 6.1.5 Version 0.9.82

- Fixed: [issue 534](#): Session parsing for Forex contracts.
- Fixed: [issue 536](#): Handle empty session field.
- Fixed: [issue 541](#): Remove superfluous closing bracket.
- Fixed: [issue 542](#): Use float size for `pnlSingle`.
- Fixed: [issue 544](#): Cancel head-time request after completion.
- Fixed: [issue 545](#): Return Trade instead of Order for `reqOpenOrders` and `reqAllOpenOrders`.

- [pull 553](#): Volume bar added.
- [pull 565](#): Typo fix.

### **6.1.6 Version 0.9.81**

- Add `ContractDetails.tradingSessions()` and `ContractDetails.liquidSessions()` to parse session times.
- Fix `IBC.on2fatimeout` command line argument for Unix.

### **6.1.7 Version 0.9.80**

- Fix `ib.reqMatchingSymbols` to handle bond contracts.

### **6.1.8 Version 0.9.79**

- Fix datetime parsing.

### **6.1.9 Version 0.9.78**

- Added `account` parameter to `ib.portfolio()`.
- Added `IBC.on2fatimeout` field.
- Removed obsolete `IBController`.
- Fixed: [issue 530](#): Use explicit timezone in requests as per new API requirement.

### **6.1.10 Version 0.9.77**

- [pull 528](#): Fixes regression in `client.py`.

### **6.1.11 Version 0.9.76**

- Fixed: [issue 525](#): For `whatIf` request treat error 110 as failure.

### **6.1.12 Version 0.9.75**

- Fixed: [issue 524](#): Use fix from Papakipos for issue with `FlexReport` downloading.

### 6.1.13 Version 0.9.74

- Fix reqContractDetails bug in combination with latest TWS.
- Update the code to comply with stricter MyPy checks.

### 6.1.14 Version 0.9.73

- [pull 523](#): Fix completedOrder parsing for new socket protocol.

### 6.1.15 Version 0.9.72

- [pull 507](#): Fixes bondContractDetails request.
- Fixed: [issue 502](#): Treat error 110 as a warning.
- Added manualOrderTime and manualCancelOrderTime for audit trails.
- Added PEG MID and PEG BEST order types.
- Added contract fields description and issuerId.
- Added IB.reqUserInfo().
- Support socket protocol version 176.

### 6.1.16 Version 0.9.71

- [pull 453](#): Added support for bidExchange and askExchange fields to Ticker.
- [pull 489](#): Watchdog.start() now returns a Future.
- Fixed: [issue 439](#): Set marketDataType directly on Ticker.
- Fixed: [issue 441](#): Add explicit timezone of None to accomodate pandas Timestamp.
- Fixed: [issue 471](#): Revised Ticker.marketPrice() calculation.
- Added minTick, bboExchange and snapshotPermissions fields to Ticker.
- Added minSize, sizeIncrement and suggestedSizeIncrement fields to ContractDetails.
- Added IB.reqHistoricalSchedule request.
- Added IB.reqSmartComponents request.
- Added Order.advancedErrorOverride field. Any advanced error message is made available from Trade.advancedError.
- Added a [recipe for integration with PyGame](#).
- Minimum required TWSAPI client protocol version is 157 now.

### 6.1.17 Version 0.9.70

- Fixed: [issue 413](#): Set the appropriate events as done on disconnect.
- Exported symbols are now static so that the VSCode/PyLance code analyzer can understand it.

### 6.1.18 Version 0.9.69

- Fixed: [issue 403](#): Change validity test for whatIfOrder response.

### 6.1.19 Version 0.9.68

- Fixed: [issue 402](#): Downloading historical ticks for crypto currencies.

### 6.1.20 Version 0.9.67

- **Crypto** security class added. To accommodate fractional crypto currency sizes, all the various `size` and `volume` fields that were of type `int` are now of type `float`.
- [pull 385](#): Get day trades remaining for next four days in `IB.accountSummary`.
- Fixed: [issue 361](#): Prevent `util.logToConsole` and `util.logToFile` from messing with the root logger.
- Fixed: [issue 370](#): Catch `asyncio.CancelledError` during connect.
- Fixed: [issue 371](#): Fix type annotation for `reqMarketRuleAsync`.
- Fixed: [issue 380](#): Reject bogus `whatIf` order response.
- Fixed: [issue 389](#): Add `TradeLogEntry.errorCode` field.

### 6.1.21 Version 0.9.66

- Fixed: [issue 360](#): Improved disconnect.
- Fixed issue with duplicate `orderId`.
- Update `Order` default values to work with the latest beta TWS/gateway.
- [pull 348](#): Added `PySide6` support.

### 6.1.22 Version 0.9.65

- Fixed: [issue 337](#).
- [pull 317](#): Update and order's `totalQuantity`, `lmtPrice`, `auxPrice` and `orderType` when the order is modified externally.
- [pull 332](#): Typo.

### 6.1.23 Version 0.9.64

- Fixed: [issue 309](#): Aggregate past fills into the Trade they belong to upon connect.
- ContFut objects are now hashable ([issue 310](#)).
- Added Watchdog.probeTimeout parameter ([issue 307](#)).

### 6.1.24 Version 0.9.63

- Fixed [issue 282](#): util.Qt() also works with the ProactorEventLoop (default on Windows) now.
- Fixed [issue 303](#): A regression in TWS 480.41+ is bypassed now to avoid IB.connect() timeouts. Request timeouts during syncing are logged as errors but will let the connect proceed.

### 6.1.25 Version 0.9.62

- IB.TimezoneTWS field added, for when the TWS timezone differs from the local system timezone ([issue 287](#)).
- IB.RaiseRequestErrors field added, can be set to True to raise RequestError when certain requests fail, instead of returning empty data ([pull 296](#)).
- IB.accountSummaryAsync() method added ([issue 267](#)).
- Watchdog.probeContract field added, to use a contract other than EURUSD for probing the data connection ([issue 298](#)).
- Ticker.rtTime added ([issue 274](#), [pull 275](#)). Please note that this timestamp appears to be mostly bogus.
- Fixed [issue 270](#): Clear ticker depth data when canceling market depth subscription.
- Fixed issue with duplicate order IDs.

### 6.1.26 Version 0.9.61

- Ticker.marketDataType added to indicate the delayed/frozen status of the reqMktData ticks.

### 6.1.27 Version 0.9.60

- IB.reqHistoricalData() has a new timeout parameter that automatically cancels the request after timing out.
- BracketOrder is iterable again.
- IB.waitOnUpdate() returns False on timeout now.
- [pull 210](#): Fix decoding of execDetails time.
- [pull 215](#): New scanner notebook added, courtesy of C. Valcarcel.
- [pull 220](#): Added readonly option for Watchdog.
- Fixed [issue 221](#): Delayed close ticks handling by Ticker.
- Fixed [issue 224](#): Added timeout for completedOrders request during connect.
- Fixed [issue 227](#): IB.MaxSyncedSubAccounts added.
- Fixed [issue 230](#): Fixed IB.reqHistogramData method.

- Fixed [issue 235](#): `Order.discretionaryAmt` is now of type `float` (was `int`).
- Fixed [issue 236](#): `ticker.updateEvent` is now fired for any change made to the ticker.
- Fixed [issue 245](#): Emit `trade.statusEvent` when order is implicitly canceled by a problem.
- You can now [sponsor the development of IB-insync!](#)

### **6.1.28 Version 0.9.59**

- PR #205 adds more typing annotations.
- `dataclasses` are now used for objects (instead of inheriting from a base `Object`). For Python 3.6.\* install it with `pip install dataclasses`

### **6.1.29 Version 0.9.58**

- PR #196 treats error 492 as a warning so that scanner results can still be used.

### **6.1.30 Version 0.9.57**

- PR #184, #185 and #186 add the new Ticker fields `rtTradeVolume`, `auctionVolume`, `auctionPrice` and `auctionImbalance`.
- PR #191 lets `util.schedule` return a handle that can be canceled.
- PR #192 adds `throttleStart` and `throttleEnd` events to the `Client`.
- PR #194 adds better JSON support for `namedtuple` objects.

### **6.1.31 Version 0.9.56**

- Fix bug #178: `Order.totalQuantity` is now `float`.

### **6.1.32 Version 0.9.55**

- Sphinx update for documentation.

### **6.1.33 Version 0.9.54**

- `ContractDetails.stockType` added.
- Fixed `Trade.filled()` for combo (BAG) contracts.
- Server version check added to make sure TWS/gateway version is at least 972.

### 6.1.34 Version 0.9.53

- Fix bug #155 (IB.commissionReportEvent not firing).
- Help editors with the code completion for Events.

### 6.1.35 Version 0.9.52

- Fix Client.exerciseOptions (bug #152).

### 6.1.36 Version 0.9.51

- Fix `ib.placeOrder` for older TWS/gateway versions.
- Better handling of unclean disconnects.

### 6.1.37 Version 0.9.50

- Fix `execDetailsEvent` regression.
- Added `readonly` argument to `ib.connect` method. Set this to `True` when the API is in read-only mode.

### 6.1.38 Version 0.9.49

- `ib.reqCompletedOrders()` request added (requires TWS/gateway  $\geq 976$ ). Completed orders are automatically synced on connect and are available from `ib.trades()`, complete with fills and commission info.
- Fixed bug #144.

### 6.1.39 Version 0.9.48

- `Ticker.halted` field added.
- `Client.reqFundamentalData` fixed.

### 6.1.40 Version 0.9.47

- `ibapi` package from IB is no longer needed, `ib_insync` handles its own socket protocol encoding and decoding now.
- Documentation moved to [readthedocs](#) as rawgit will cease operation later this year.
- Blocking requests will now raise `ConnectionError` on a connection failure. This also goes for `util.run`, `util.timeRange`, etc.

### 6.1.41 Version 0.9.46

- Event class has been replaced with the one from `eventkit`.
- Event-driven bar construction from ticks added (via `Ticker.updateEvent`)
- Fixed bug #136.
- Default request throttling is now 45 requests/s for compatibility with TWS/gateway 974 and higher.

### 6.1.42 Version 0.9.45

- `Event.merge()` added.
- `TagValue` serialization fixed.

### 6.1.43 Version 0.9.44

- `Event.any()` and `Event.all()` added.
- Ticker fields added: `tradeCount`, `tradeRate`, `volumeRate`, `avOptionVolume`, `markPrice`, `histVolatility`, `impliedVolatility`, `rtHistVolatility` and `indexFuturePremium`.
- Parse `ticker.fundamentalRatios` into `FundamentalRatios` object.
- `util.timeRangeAsync()` and `waitUntilAsync()` added.
- `ib.pendingTickersEvent` now emits a set of Tickers instead of a list.
- Tick handling has been streamlined.
- For harvesting tick data, an imperative code style with a `waitOnUpdate` loop should not be used anymore!

### 6.1.44 Version 0.9.43

- Fixed issue #132.
- `Event.aiter()` added, all events can now be used as asynchronous iterators.
- `Event.wait()` added, all events are now also awaitable.
- Decreased default throttling to 95 requests per 2 sec.

### 6.1.45 Version 0.9.42

- `Ticker.shortableShares` added (for use with generic tick 236).
- `ib.reqAllOpenOrders()` request added.
- `tickByTick` subscription will update ticker's bid, ask, last, etc.
- Drop redundant bid/ask ticks from `reqMktData`.
- Fixed occasional "Group name cannot be null" error message on connect.
- Watchdog code rewritten to not need `util.patchAsyncio`.
- `Watchdog.start()` is no longer blocking.



### 6.1.46 Version 0.9.41

- Fixed bug #117.
- Fixed order modifications with TWS/gateway 974.

### 6.1.47 Version 0.9.40

- `Ticker.fundamentalRatios` added (for use with generic tick 258).
- Fixed `reqHistoricalTicks` with MIDPOINT.

### 6.1.48 Version 0.9.39

- Handle partially filled dividend data.
- Use `secType='WAR'` for warrants.

### 6.1.49 Version 0.9.38

- `ibapi v97.4` is now required.
- fixed `tickByTick` wrappers.

### 6.1.50 Version 0.9.37

- Backward compatibility with older `ibapi` restored.

### 6.1.51 Version 0.9.36

- Compatibility with `ibapi v974`.
- `Client.setConnectOptions()` added (for PACEAPI).

### 6.1.52 Version 0.9.35

- `Ticker.hasBidAsk()` added.
- `IB.newsBulletinEvent` added.
- Various small fixes.

### 6.1.53 Version 0.9.34

- Old event system (`ib.setCallback`) removed.
- Compatibility fix with previous `ibapi` version.

### 6.1.54 Version 0.9.33

- Market scanner subscription improved.
- `IB.scannerDataEvent` now emits the full list of `ScanData`.
- `ScanDataList` added.

### 6.1.55 Version 0.9.32

- Autocompletion with Jedi plugin as used in Spyder and VS Code working again.

### 6.1.56 Version 0.9.31

- Request results will return specialized contract types (like `Stock`) instead of generic `Contract`.
- `IB.scannerDataEvent` added.
- `ContractDetails` field `summary` renamed to `contract`.
- `isSmartDepth` parameter added for `reqMktDepth`.
- Event loop nesting is now handled by the [nest\\_asyncio project](#).
- `util.useQt` is rewritten so that it can be used with any asyncio event loop, with support for both PyQt5 and PySide2. It does not use `quamash` anymore.
- Various fixes, extensive documentation overhaul and flake8-compliant code formatting.

### 6.1.57 Version 0.9.30

- `Watchdog.stop()` will not trigger restart now.
- Fixed bug #93.

### 6.1.58 Version 0.9.29

- `util.patchAsyncio()` updated for Python 3.7.

### 6.1.59 Version 0.9.28

- `IB.RequestTimeout` added.
- `util.schedule()` accepts tz-aware datetimes now.
- Let `client.disconnect()` complete when no event loop is running.

### 6.1.60 Version 0.9.27

- Fixed bug #77.

### 6.1.61 Version 0.9.26

- PR #74 merged (`ib.reqCurrentTime()` method added).
- Fixed bug with order error handling.

### 6.1.62 Version 0.9.25

- Default throttling rate now compatible with `reqTickers`.
- Fixed issue with `ib.waitOnUpdate()` in combination. with `ib.pendingTickersEvent`.
- Added timeout parameter for `ib.waitOnUpdate()`.

### 6.1.63 Version 0.9.24

- `ticker.futuresOpenInterest` added.
- `execution.time` was string, is now parsed to UTC datetime.
- `ib.reqMarketRule()` request added.

### 6.1.64 Version 0.9.23

- Compatibility with Tornado 5 as used in new Jupyter notebook server.

### 6.1.65 Version 0.9.22

- updated `ib.reqNewsArticle` and `ib.reqHistoricalNews` to `ibapi v9.73.07`.

### 6.1.66 Version 0.9.21

- updated `ib.reqTickByTickData()` signature to `ibapi v9.73.07` while keeping backward compatibility.

### 6.1.67 Version 0.9.20

- Fixed watchdog bug.

### **6.1.68 Version 0.9.19**

- Don't overwrite `exchange='SMART'` in `qualifyContracts`.

### **6.1.69 Version 0.9.18**

- Merged PR #65 (Fix misnamed event).

### **6.1.70 Version 0.9.17**

- New IB events `disconnectedEvent`, `newOrderEvent`, `orderModifyEvent` and `cancelOrderEvent`.
- Watchdog improvements.

### **6.1.71 Version 0.9.16**

- New event system that will supersede `IB.setCallback()`.
- Notebooks updated to use events.
- Watchdog must now be given an IB instance.

### **6.1.72 Version 0.9.15**

- Fixed bug in default order conditions.
- Fixed regression from v0.9.13 in `placeOrder`.

### **6.1.73 Version 0.9.14**

- Fixed `orderStatus` callback regression.

### **6.1.74 Version 0.9.13**

- Log handling improvements.
- Client with `clientId=0` can now manage manual TWS orders.
- Client with master `clientId` can now monitor manual TWS orders.

### **6.1.75 Version 0.9.12**

- Run IBC and `IBController` directly instead of via shell.

### 6.1.76 Version 0.9.11

- Fixed bug when collecting ticks using `ib.waitOnUpdate()`.
- Added `ContFuture` class (continuous futures).
- Added `Ticker.midpoint()`.

### 6.1.77 Version 0.9.10

- `ib.accountValues()` fixed for use with multiple accounts.

### 6.1.78 Version 0.9.9

- Fixed issue #57

### 6.1.79 Version 0.9.8

- Fix for `ib.reqPnLSingle()`.

### 6.1.80 Version 0.9.7

- Profit and Loss (PnL) functionality added.

### 6.1.81 Version 0.9.6

- IBC added.
- PR #53 (delayed greeks) merged.
- `Ticker.futuresOpenInterest` field removed.

### 6.1.82 Version 0.9.5

- Fixed canceling bar and tick subscriptions.

### 6.1.83 Version 0.9.4

- Fixed issue #49.

### 6.1.84 Version 0.9.3

- Watchdog class added.
- `ib.setTimeout()` added.
- `Ticker.dividends` added for use with `genericTickList` 456.
- Errors and warnings will now log the contract they apply to.
- `IB.error()` callback signature changed to include contract.
- Fix for issue #44.

### 6.1.85 Version 0.9.2

- Historical ticks and realtime bars now return time in UTC.

### 6.1.86 Version 0.9.1

- `IBController` added.
- `openOrder` callback added.
- default arguments for `ib.connect()` and `ib.reqMktData()`.

### 6.1.87 Version 0.9.0

- minimum API version is v9.73.06.
- `tickByTick` support.
- automatic request throttling.
- `ib.accountValues()` now works for multiple accounts.
- `AccountValue.modelCode` added.
- `Ticker.rtVolume` added.

## 6.2 0.8

### 6.2.1 Version 0.8.17

- workaround for IBAPI v9.73.06 for `Contract.lastTradeDateOrContractMonth` format.

### 6.2.2 Version 0.8.16

- `util.tree()` method added.
- `error` callback signature changed to `(reqId, errorCode, errorString)`.
- `accountValue` and `accountSummary` callbacks added.

### 6.2.3 Version 0.8.15

- `util.useQt()` fixed for use with Windows.

### 6.2.4 Version 0.8.14

- Fix for `ib.schedule()`.

### 6.2.5 Version 0.8.13

- Import order conditions into `ib_insync` namespace.
- `util.useQtAlt()` added for using nested event loops on Windows with Qtl
- `ib.schedule()` added.

### 6.2.6 Version 0.8.12

- Fixed conditional orders.

### 6.2.7 Version 0.8.11

- `FlexReport` added.

### 6.2.8 Version 0.8.10

- Fixed issue #22.

### 6.2.9 Version 0.8.9

- `Ticker.vwap` field added (for use with generic tick 233).
- Client with master `clientId` can now monitor orders and trades of other clients.

### 6.2.10 Version 0.8.8

- `barUpdate` event now used also for `reqRealTimeBars` responses
- `reqRealTimeBars` will return `RealTimeBarList` instead of `list`.
- realtime bars example added to bar data notebook.
- fixed event handling bug in `Wrapper.execDetails`.

### 6.2.11 Version 0.8.7

- `BarDataList` now used with `reqHistoricalData`; it also stores the request parameters.
- updated the typing annotations.
- added `barUpdate` event to `IB`.
- bar- and tick-data notebooks updated to use callbacks for realtime data.

### 6.2.12 Version 0.8.6

- `ticker.marketPrice` adjusted to ignore price of -1.
- `ticker.avVolume` handling fixed.

### 6.2.13 Version 0.8.5

- `realtimeBar` wrapper fix.
- context manager for `IB` and `IB.connect()`.

### 6.2.14 Version 0.8.4

- compatibility with upcoming `ibapi` changes.
- added `error` event to `IB`.
- notebooks updated to use `loopUntil`.
- small fixes and performance improvements.

### 6.2.15 Version 0.8.3

- new `IB.reqHistoricalTicks()` API method.
- new `IB.loopUntil()` method.
- fixed issues #4, #6, #7.



### 6.2.16 Version 0.8.2

- fixed swapped `ticker.putOpenInterest` vs `ticker.callOpenInterest`.

### 6.2.17 Version 0.8.1

- fixed `wrapper.tickSize` regression.

### 6.2.18 Version 0.8.0

- support for realtime bars and `keepUpToDate` for historical bars
- added option greeks to `Ticker`.
- new `IB.waitUntil()` and `IB.timeRange()` scheduling methods.
- notebooks no longer depend on PyQt5 for live updates.
- notebooks can be run in one go ('run all').
- tick handling bypasses `ibapi` decoder for more efficiency.

## 6.3 0.7

### 6.3.1 Version 0.7.3

- `IB.whatIfOrder()` added.
- Added detection and warning about common setup problems.

### 6.3.2 Version 0.7.2

- Removed import from `ipykernel`.

### 6.3.3 Version 0.7.1

- Removed dependencies for installing via `pip`.

### 6.3.4 Version 0.7.0

- added lots of request methods.
- order book (DOM) added.
- notebooks updated.

## 6.4 0.6

### 6.4.1 Version 0.6.1

- Added UTC timezone to some timestamps.
- Fixed issue #1.

### 6.4.2 Version 0.6.0

- Initial release.

## LINKS

- [Interactive Brokers](#)
- [Interactive Brokers Python API](#)
- [TWSAPI documentation](#)
- [TWSAPI user group](#)
- [IB-insync user group](#)
- [Dmitry's TWS API FAQ](#)
- [IBC](#) for hands-free operation of TWS or gateway



## INTRODUCTION

The goal of the IB-insync library is to make working with the [Trader Workstation API](#) from Interactive Brokers as easy as possible.

The main features are:

- An easy to use linear style of programming;
- An [IB component](#) that automatically keeps in sync with the TWS or IB Gateway application;
- A fully asynchronous framework based on [asyncio](#) and [eventkit](#) for advanced users;
- Interactive operation with live data in Jupyter notebooks.

Be sure to take a look at the [notebooks](#), the [recipes](#) and the [API docs](#).

## 8.1 Installation

```
pip install ib_insync
```

Requirements:

- Python 3.6 or higher;
- A running TWS or IB Gateway application (version 1023 or higher). Make sure the [API port is enabled](#) and 'Download open orders on connection' is checked.

The ibapi package from IB is not needed.

## 8.2 Example

This is a complete script to download historical data:

```
from ib_insync import *
# util.startLoop() # uncomment this line when in a notebook

ib = IB()
ib.connect('127.0.0.1', 7497, clientId=1)

contract = Forex('EURUSD')
bars = ib.reqHistoricalData(
    contract, endDateTime='', durationStr='30 D',
```

(continues on next page)

(continued from previous page)

```

barSizeSetting='1 hour', whatToShow='MIDPOINT', useRTH=True)

# convert to pandas dataframe (pandas needs to be installed):
df = util.df(bars)
print(df)

```

Output:

	date	open	high	low	close	volume	\
0	2019-11-19 23:15:00	1.107875	1.108050	1.107725	1.107825	-1	
1	2019-11-20 00:00:00	1.107825	1.107925	1.107675	1.107825	-1	
2	2019-11-20 01:00:00	1.107825	1.107975	1.107675	1.107875	-1	
3	2019-11-20 02:00:00	1.107875	1.107975	1.107025	1.107225	-1	
4	2019-11-20 03:00:00	1.107225	1.107725	1.107025	1.107525	-1	
...	...	...	...	...	...	...	
705	2020-01-02 14:00:00	1.119325	1.119675	1.119075	1.119225	-1	

## 8.3 Documentation

The complete API documentation.

Changelog.

## 8.4 Discussion

The [insync user group](#) is the place to discuss IB-insync and anything related to it.

## 8.5 Disclaimer

The software is provided on the conditions of the simplified BSD license.

This project is not affiliated with Interactive Brokers Group, Inc.'s.

Good luck and enjoy.

**author**

Ewald de Wit <[ewald.de.wit@gmail.com](mailto:ewald.de.wit@gmail.com)>

## PYTHON MODULE INDEX

### i

- `ib_insync.client`, [30](#)
- `ib_insync.contract`, [52](#)
- `ib_insync.flexreport`, [98](#)
- `ib_insync.ib`, [5](#)
- `ib_insync.objects`, [75](#)
- `ib_insync.order`, [35](#)
- `ib_insync.ticker`, [68](#)
- `ib_insync.util`, [94](#)





## A

- abovePrice (*ib\_insync.objects.ScannerSubscription attribute*), 75
- aboveVolume (*ib\_insync.objects.ScannerSubscription attribute*), 75
- account (*ib\_insync.ibcontroller.Watchdog attribute*), 102
- account (*ib\_insync.objects.AccountValue attribute*), 88
- account (*ib\_insync.objects.PnL attribute*), 84
- account (*ib\_insync.objects.PnLSingle attribute*), 85
- account (*ib\_insync.objects.PortfolioItem attribute*), 91
- account (*ib\_insync.objects.Position attribute*), 91
- account (*ib\_insync.order.Order attribute*), 39
- accountID (*ib\_insync.objects.FamilyCode attribute*), 93
- accountSummary() (*ib\_insync.ib.IB method*), 10
- accountSummaryAsync() (*ib\_insync.ib.IB method*), 27
- AccountValue (*class in ib\_insync.objects*), 88
- accountValues() (*ib\_insync.ib.IB method*), 9
- acctCode (*ib\_insync.objects.ExecutionFilter attribute*), 79
- acctNumber (*ib\_insync.objects.Execution attribute*), 77
- action (*ib\_insync.contract.ComboLeg attribute*), 64
- action (*ib\_insync.order.Order attribute*), 37
- activeStartTime (*ib\_insync.order.Order attribute*), 37
- ActiveStates (*ib\_insync.order.OrderStatus attribute*), 44
- activeStopTime (*ib\_insync.order.Order attribute*), 37
- adjustableTrailingUnit (*ib\_insync.order.Order attribute*), 40
- adjustedOrderType (*ib\_insync.order.Order attribute*), 40
- adjustedStopLimitPrice (*ib\_insync.order.Order attribute*), 40
- adjustedStopPrice (*ib\_insync.order.Order attribute*), 40
- adjustedTrailingAmount (*ib\_insync.order.Order attribute*), 40
- advancedError (*ib\_insync.order.Trade attribute*), 47
- advancedErrorOverride (*ib\_insync.order.Order attribute*), 41
- aggGroup (*ib\_insync.contract.ContractDetails attribute*), 66
- aggGroup (*ib\_insync.objects.DepthMktDataDescription attribute*), 84
- algoId (*ib\_insync.order.Order attribute*), 39
- algoParams (*ib\_insync.order.Order attribute*), 39
- algoStrategy (*ib\_insync.order.Order attribute*), 39
- allOrNone (*ib\_insync.order.Order attribute*), 37
- allowCtrlC() (*in module ib\_insync.util*), 95
- And() (*ib\_insync.order.OrderCondition method*), 48
- ApiCancelled (*ib\_insync.order.OrderStatus attribute*), 44
- ApiPending (*ib\_insync.order.OrderStatus attribute*), 44
- appStartupTime (*ib\_insync.ibcontroller.Watchdog attribute*), 102
- appTimeout (*ib\_insync.ibcontroller.Watchdog attribute*), 102
- articleId (*ib\_insync.objects.HistoricalNews attribute*), 92
- articleId (*ib\_insync.objects.NewsTick attribute*), 92
- articleText (*ib\_insync.objects.NewsArticle attribute*), 92
- articleType (*ib\_insync.objects.NewsArticle attribute*), 92
- ask (*ib\_insync.ticker.Ticker attribute*), 69
- askExchange (*ib\_insync.ticker.Ticker attribute*), 69
- askGreeks (*ib\_insync.ticker.Ticker attribute*), 71
- askPastHigh (*ib\_insync.objects.TickAttribBidAsk attribute*), 82
- askPrice (*ib\_insync.objects.TickByTickBidAsk attribute*), 89
- asks() (*ib\_insync.ticker.TickerUpdateEvent method*), 72
- askSize (*ib\_insync.objects.TickByTickBidAsk attribute*), 90
- askSize (*ib\_insync.ticker.Ticker attribute*), 69
- askYield (*ib\_insync.ticker.Ticker attribute*), 70
- auctionImbalance (*ib\_insync.ticker.Ticker attribute*), 71
- auctionPrice (*ib\_insync.ticker.Ticker attribute*), 71
- auctionStrategy (*ib\_insync.order.Order attribute*), 38
- auctionVolume (*ib\_insync.ticker.Ticker attribute*), 71
- autoCancelDate (*ib\_insync.order.Order attribute*), 40
- autoCancelParent (*ib\_insync.order.Order attribute*), 41

`auxPrice` (*ib\_insync.order.Order* attribute), 37  
`average` (*ib\_insync.objects.BarData* attribute), 80  
`averageCost` (*ib\_insync.objects.PortfolioItem* attribute), 90  
`averageOptionVolumeAbove` (*ib\_insync.objects.ScannerSubscription* attribute), 76  
`avgCost` (*ib\_insync.objects.Position* attribute), 91  
`avgFillPrice` (*ib\_insync.order.OrderStatus* attribute), 43  
`avgPrice` (*ib\_insync.objects.Execution* attribute), 77  
`avOptionVolume` (*ib\_insync.ticker.Ticker* attribute), 71  
`avVolume` (*ib\_insync.ticker.Ticker* attribute), 70

## B

`Bag` (class in *ib\_insync.contract*), 62  
`Bar` (class in *ib\_insync.ticker*), 74  
`barCount` (*ib\_insync.objects.BarData* attribute), 80  
`BarData` (class in *ib\_insync.objects*), 79  
`BarDataList` (class in *ib\_insync.objects*), 93  
`BarList` (class in *ib\_insync.ticker*), 74  
`barplot()` (in module *ib\_insync.util*), 95  
`bars` (*ib\_insync.ticker.TickBars* attribute), 74  
`bars` (*ib\_insync.ticker.TimeBars* attribute), 74  
`bars` (*ib\_insync.ticker.VolumeBars* attribute), 75  
`barSize` (*ib\_insync.objects.RealTimeBarList* attribute), 94  
`barSizeSetting` (*ib\_insync.objects.BarDataList* attribute), 93  
`basisPoints` (*ib\_insync.order.Order* attribute), 39  
`basisPointsType` (*ib\_insync.order.Order* attribute), 39  
`bboExchange` (*ib\_insync.ticker.Ticker* attribute), 71  
`belowPrice` (*ib\_insync.objects.ScannerSubscription* attribute), 75  
`benchmark` (*ib\_insync.contract.ScanData* attribute), 68  
`bid` (*ib\_insync.ticker.Ticker* attribute), 69  
`bidasks()` (*ib\_insync.ticker.TickerUpdateEvent* method), 73  
`bidExchange` (*ib\_insync.ticker.Ticker* attribute), 69  
`bidGreeks` (*ib\_insync.ticker.Ticker* attribute), 71  
`bidPastLow` (*ib\_insync.objects.TickAttribBidAsk* attribute), 81  
`bidPrice` (*ib\_insync.objects.TickByTickBidAsk* attribute), 89  
`bids()` (*ib\_insync.ticker.TickerUpdateEvent* method), 72  
`bidSize` (*ib\_insync.objects.TickByTickBidAsk* attribute), 89  
`bidSize` (*ib\_insync.ticker.Ticker* attribute), 69  
`bidYield` (*ib\_insync.ticker.Ticker* attribute), 70  
`bitNumber` (*ib\_insync.objects.SmartComponent* attribute), 93  
`blockOrder` (*ib\_insync.order.Order* attribute), 37  
`Bond` (class in *ib\_insync.contract*), 60

`bondType` (*ib\_insync.contract.ContractDetails* attribute), 66  
`BracketOrder` (class in *ib\_insync.order*), 47  
`bracketOrder()` (*ib\_insync.ib.IB* method), 12

## C

`calculateImpliedVolatility()` (*ib\_insync.client.Client* method), 33  
`calculateImpliedVolatility()` (*ib\_insync.ib.IB* method), 23  
`calculateImpliedVolatilityAsync()` (*ib\_insync.ib.IB* method), 29  
`calculateOptionPrice()` (*ib\_insync.client.Client* method), 33  
`calculateOptionPrice()` (*ib\_insync.ib.IB* method), 23  
`calculateOptionPriceAsync()` (*ib\_insync.ib.IB* method), 29  
`callable` (*ib\_insync.contract.ContractDetails* attribute), 66  
`callOpenInterest` (*ib\_insync.ticker.Ticker* attribute), 71  
`callVolume` (*ib\_insync.ticker.Ticker* attribute), 71  
`canAutoExecute` (*ib\_insync.objects.TickAttrib* attribute), 81  
`cancelAccountSummary()` (*ib\_insync.client.Client* method), 33  
`cancelAccountUpdatesMulti()` (*ib\_insync.client.Client* method), 34  
`cancelCalculateImpliedVolatility()` (*ib\_insync.client.Client* method), 33  
`cancelCalculateOptionPrice()` (*ib\_insync.client.Client* method), 33  
`cancelFundamentalData()` (*ib\_insync.client.Client* method), 33  
`cancelHeadTimeStamp()` (*ib\_insync.client.Client* method), 34  
`cancelHistogramData()` (*ib\_insync.client.Client* method), 34  
`cancelHistoricalData()` (*ib\_insync.client.Client* method), 33  
`cancelHistoricalData()` (*ib\_insync.ib.IB* method), 18  
`Cancelled` (*ib\_insync.order.OrderStatus* attribute), 44  
`cancelMktData()` (*ib\_insync.client.Client* method), 32  
`cancelMktData()` (*ib\_insync.ib.IB* method), 20  
`cancelMktDepth()` (*ib\_insync.client.Client* method), 32  
`cancelMktDepth()` (*ib\_insync.ib.IB* method), 21  
`cancelNewsBulletins()` (*ib\_insync.client.Client* method), 32  
`cancelNewsBulletins()` (*ib\_insync.ib.IB* method), 25  
`cancelOrder()` (*ib\_insync.client.Client* method), 32  
`cancelOrder()` (*ib\_insync.ib.IB* method), 13  
`cancelPnL()` (*ib\_insync.client.Client* method), 34

cancelPnL() (*ib\_insync.ib.IB method*), 15  
cancelPnLSingle() (*ib\_insync.client.Client method*), 34  
cancelPnLSingle() (*ib\_insync.ib.IB method*), 16  
cancelPositions() (*ib\_insync.client.Client method*), 33  
cancelPositionsMulti() (*ib\_insync.client.Client method*), 33  
cancelRealTimeBars() (*ib\_insync.client.Client method*), 33  
cancelRealTimeBars() (*ib\_insync.ib.IB method*), 17  
cancelScannerSubscription() (*ib\_insync.client.Client method*), 33  
cancelScannerSubscription() (*ib\_insync.ib.IB method*), 22  
cancelTickByTickData() (*ib\_insync.client.Client method*), 34  
cancelTickByTickData() (*ib\_insync.ib.IB method*), 20  
cancelWshEventData() (*ib\_insync.client.Client method*), 34  
cancelWshEventData() (*ib\_insync.ib.IB method*), 26  
cancelWshMetaData() (*ib\_insync.client.Client method*), 34  
cancelWshMetaData() (*ib\_insync.ib.IB method*), 25  
cashQty (*ib\_insync.order.Order attribute*), 40  
category (*ib\_insync.contract.ContractDetails attribute*), 65  
CFD (*class in ib\_insync.contract*), 59  
changePercent (*ib\_insync.order.PercentChangeCondition attribute*), 51  
chartOptions (*ib\_insync.objects.BarDataList attribute*), 93  
clearingAccount (*ib\_insync.order.Order attribute*), 39  
clearingIntent (*ib\_insync.order.Order attribute*), 39  
Client (*class in ib\_insync.client*), 30  
clientId (*ib\_insync.ibcontroller.Watchdog attribute*), 102  
clientId (*ib\_insync.objects.Execution attribute*), 77  
clientId (*ib\_insync.objects.ExecutionFilter attribute*), 79  
clientId (*ib\_insync.order.Order attribute*), 37  
clientId (*ib\_insync.order.OrderStatus attribute*), 44  
close (*ib\_insync.objects.BarData attribute*), 80  
close (*ib\_insync.objects.RealTimeBar attribute*), 80  
close (*ib\_insync.ticker.Bar attribute*), 74  
close (*ib\_insync.ticker.Ticker attribute*), 70  
code (*ib\_insync.objects.NewsProvider attribute*), 83  
ComboLeg (*class in ib\_insync.contract*), 63  
comboLegs (*ib\_insync.contract.Contract attribute*), 54  
comboLegsDescrip (*ib\_insync.contract.Contract attribute*), 54  
commission (*ib\_insync.objects.CommissionReport attribute*), 78  
commission (*ib\_insync.order.OrderState attribute*), 45  
commissionCurrency (*ib\_insync.order.OrderState attribute*), 45  
CommissionReport (*class in ib\_insync.objects*), 78  
commissionReport (*ib\_insync.objects.Fill attribute*), 91  
Commodity (*class in ib\_insync.contract*), 59  
competeAgainstBestOffset (*ib\_insync.order.Order attribute*), 41  
completedStatus (*ib\_insync.order.OrderState attribute*), 45  
completedTime (*ib\_insync.order.OrderState attribute*), 45  
conditions (*ib\_insync.order.Order attribute*), 40  
conditionsCancelOrder (*ib\_insync.order.Order attribute*), 40  
conditionsIgnoreRth (*ib\_insync.order.Order attribute*), 40  
condType (*ib\_insync.order.ExecutionCondition attribute*), 50  
condType (*ib\_insync.order.MarginCondition attribute*), 49  
condType (*ib\_insync.order.PercentChangeCondition attribute*), 51  
condType (*ib\_insync.order.PriceCondition attribute*), 48  
condType (*ib\_insync.order.TimeCondition attribute*), 49  
condType (*ib\_insync.order.VolumeCondition attribute*), 51  
conId (*ib\_insync.contract.ComboLeg attribute*), 63  
conId (*ib\_insync.contract.Contract attribute*), 54  
conId (*ib\_insync.contract.DeltaNeutralContract attribute*), 64  
conId (*ib\_insync.objects.PnLSingle attribute*), 85  
conId (*ib\_insync.objects.WshEventData attribute*), 87  
conId (*ib\_insync.order.PercentChangeCondition attribute*), 51  
conId (*ib\_insync.order.PriceCondition attribute*), 48  
conId (*ib\_insync.order.VolumeCondition attribute*), 51  
conjunction (*ib\_insync.order.ExecutionCondition attribute*), 50  
conjunction (*ib\_insync.order.MarginCondition attribute*), 49  
conjunction (*ib\_insync.order.PercentChangeCondition attribute*), 51  
conjunction (*ib\_insync.order.PriceCondition attribute*), 48  
conjunction (*ib\_insync.order.TimeCondition attribute*), 49  
conjunction (*ib\_insync.order.VolumeCondition attribute*), 51  
connect() (*ib\_insync.client.Client method*), 31  
connect() (*ib\_insync.ib.IB method*), 7  
connectAsync() (*ib\_insync.client.Client method*), 32  
connectAsync() (*ib\_insync.ib.IB method*), 27  
CONNECTED (*ib\_insync.client.Client attribute*), 31

- CONNECTING (*ib\_insync.client.Client* attribute), 31
- ConnectionStats (*class* in *ib\_insync.objects*), 93
- connectionStats() (*ib\_insync.client.Client* method), 31
- connectTimeout (*ib\_insync.ibcontroller.Watchdog* attribute), 102
- ContFuture (*class* in *ib\_insync.contract*), 57
- continuousUpdate (*ib\_insync.order.Order* attribute), 39
- Contract (*class* in *ib\_insync.contract*), 52
- contract (*ib\_insync.contract.ContractDescription* attribute), 67
- contract (*ib\_insync.contract.ContractDetails* attribute), 65
- contract (*ib\_insync.objects.BarDataList* attribute), 93
- contract (*ib\_insync.objects.Fill* attribute), 91
- contract (*ib\_insync.objects.PortfolioItem* attribute), 90
- contract (*ib\_insync.objects.Position* attribute), 91
- contract (*ib\_insync.objects.RealTimeBarList* attribute), 94
- contract (*ib\_insync.order.Trade* attribute), 46
- contract (*ib\_insync.ticker.Ticker* attribute), 69
- ContractDescription (*class* in *ib\_insync.contract*), 67
- ContractDetails (*class* in *ib\_insync.contract*), 65
- contractDetails (*ib\_insync.contract.ScanData* attribute), 68
- contractMonth (*ib\_insync.contract.ContractDetails* attribute), 65
- controller (*ib\_insync.ibcontroller.Watchdog* attribute), 102
- convertible (*ib\_insync.contract.ContractDetails* attribute), 66
- count (*ib\_insync.objects.HistogramData* attribute), 83
- count (*ib\_insync.objects.RealTimeBar* attribute), 80
- count (*ib\_insync.ticker.Bar* attribute), 74
- coupon (*ib\_insync.contract.ContractDetails* attribute), 66
- couponRateAbove (*ib\_insync.objects.ScannerSubscription* attribute), 76
- couponRateBelow (*ib\_insync.objects.ScannerSubscription* attribute), 76
- couponType (*ib\_insync.contract.ContractDetails* attribute), 66
- create() (*ib\_insync.contract.Contract* static method), 54
- createClass() (*ib\_insync.order.OrderCondition* static method), 48
- Crypto (*class* in *ib\_insync.contract*), 63
- cumQty (*ib\_insync.objects.Execution* attribute), 77
- currency (*ib\_insync.contract.Contract* attribute), 54
- currency (*ib\_insync.objects.AccountValue* attribute), 88
- currency (*ib\_insync.objects.CommissionReport* attribute), 78
- cusip (*ib\_insync.contract.ContractDetails* attribute), 66
- D**
- dailyPnL (*ib\_insync.objects.PnL* attribute), 84
- dailyPnL (*ib\_insync.objects.PnLSingle* attribute), 86
- data (*ib\_insync.flexreport.FlexReport* attribute), 98
- dataclassAsDict() (*in module ib\_insync.util*), 95
- dataclassAsTuple() (*in module ib\_insync.util*), 95
- dataclassNonDefaults() (*in module ib\_insync.util*), 95
- dataclassRepr() (*in module ib\_insync.util*), 95
- dataclassUpdate() (*in module ib\_insync.util*), 95
- date (*ib\_insync.objects.BarData* attribute), 79
- delta (*ib\_insync.contract.DeltaNeutralContract* attribute), 64
- delta (*ib\_insync.objects.OptionComputation* attribute), 91
- delta (*ib\_insync.order.Order* attribute), 38
- deltaNeutralAuxPrice (*ib\_insync.order.Order* attribute), 38
- deltaNeutralClearingAccount (*ib\_insync.order.Order* attribute), 38
- deltaNeutralClearingIntent (*ib\_insync.order.Order* attribute), 39
- deltaNeutralConId (*ib\_insync.order.Order* attribute), 38
- DeltaNeutralContract (*class* in *ib\_insync.contract*), 64
- deltaNeutralContract (*ib\_insync.contract.Contract* attribute), 54
- deltaNeutralDesignatedLocation (*ib\_insync.order.Order* attribute), 39
- deltaNeutralOpenClose (*ib\_insync.order.Order* attribute), 39
- deltaNeutralOrderType (*ib\_insync.order.Order* attribute), 38
- deltaNeutralSettlingFirm (*ib\_insync.order.Order* attribute), 38
- deltaNeutralShortSale (*ib\_insync.order.Order* attribute), 39
- deltaNeutralShortSaleSlot (*ib\_insync.order.Order* attribute), 39
- DepthMktDataDescription (*class* in *ib\_insync.objects*), 83
- derivativeSecTypes (*ib\_insync.contract.ContractDescription* attribute), 67
- descAppend (*ib\_insync.contract.ContractDetails* attribute), 66
- description (*ib\_insync.contract.Contract* attribute), 54
- designatedLocation (*ib\_insync.contract.ComboLeg* attribute), 64
- designatedLocation (*ib\_insync.order.Order* attribute), 38
- df() (*ib\_insync.flexreport.FlexReport* method), 98
- df() (*in module ib\_insync.util*), 94
- dict() (*ib\_insync.contract.Bag* method), 62



dict() (*ib\_insync.contract.Bond* method), 60  
 dict() (*ib\_insync.contract.CFD* method), 59  
 dict() (*ib\_insync.contract.ComboLeg* method), 64  
 dict() (*ib\_insync.contract.Commodity* method), 60  
 dict() (*ib\_insync.contract.ContFuture* method), 57  
 dict() (*ib\_insync.contract.Contract* method), 54  
 dict() (*ib\_insync.contract.ContractDescription* method), 67  
 dict() (*ib\_insync.contract.ContractDetails* method), 67  
 dict() (*ib\_insync.contract.Crypto* method), 63  
 dict() (*ib\_insync.contract.DeltaNeutralContract* method), 64  
 dict() (*ib\_insync.contract.Forex* method), 58  
 dict() (*ib\_insync.contract.Future* method), 57  
 dict() (*ib\_insync.contract.FuturesOption* method), 61  
 dict() (*ib\_insync.contract.Index* method), 58  
 dict() (*ib\_insync.contract.MutualFund* method), 61  
 dict() (*ib\_insync.contract.Option* method), 56  
 dict() (*ib\_insync.contract.ScanData* method), 68  
 dict() (*ib\_insync.contract.Stock* method), 55  
 dict() (*ib\_insync.contract.Warrant* method), 62  
 dict() (*ib\_insync.ibcontroller.IBC* method), 100  
 dict() (*ib\_insync.ibcontroller.Watchdog* method), 102  
 dict() (*ib\_insync.objects.BarData* method), 80  
 dict() (*ib\_insync.objects.CommissionReport* method), 78  
 dict() (*ib\_insync.objects.DepthMktDataDescription* method), 84  
 dict() (*ib\_insync.objects.Execution* method), 77  
 dict() (*ib\_insync.objects.ExecutionFilter* method), 79  
 dict() (*ib\_insync.objects.HistogramData* method), 83  
 dict() (*ib\_insync.objects.HistoricalSchedule* method), 87  
 dict() (*ib\_insync.objects.HistoricalSession* method), 86  
 dict() (*ib\_insync.objects.NewsProvider* method), 83  
 dict() (*ib\_insync.objects.PnL* method), 84  
 dict() (*ib\_insync.objects.PnLSingle* method), 86  
 dict() (*ib\_insync.objects.RealTimeBar* method), 80  
 dict() (*ib\_insync.objects.ScannerSubscription* method), 76  
 dict() (*ib\_insync.objects.SoftDollarTier* method), 76  
 dict() (*ib\_insync.objects.TickAttrib* method), 81  
 dict() (*ib\_insync.objects.TickAttribBidAsk* method), 82  
 dict() (*ib\_insync.objects.TickAttribLast* method), 82  
 dict() (*ib\_insync.objects.TradeLogEntry* method), 85  
 dict() (*ib\_insync.objects.WshEventData* method), 88  
 dict() (*ib\_insync.order.ExecutionCondition* method), 50  
 dict() (*ib\_insync.order.LimitOrder* method), 42  
 dict() (*ib\_insync.order.MarginCondition* method), 50  
 dict() (*ib\_insync.order.MarketOrder* method), 42  
 dict() (*ib\_insync.order.Order* method), 41  
 dict() (*ib\_insync.order.OrderComboLeg* method), 46  
 dict() (*ib\_insync.order.OrderCondition* method), 48  
 dict() (*ib\_insync.order.OrderState* method), 45  
 dict() (*ib\_insync.order.OrderStatus* method), 44  
 dict() (*ib\_insync.order.PercentChangeCondition* method), 52  
 dict() (*ib\_insync.order.PriceCondition* method), 48  
 dict() (*ib\_insync.order.StopLimitOrder* method), 43  
 dict() (*ib\_insync.order.StopOrder* method), 42  
 dict() (*ib\_insync.order.TimeCondition* method), 49  
 dict() (*ib\_insync.order.Trade* method), 47  
 dict() (*ib\_insync.order.VolumeCondition* method), 51  
 dict() (*ib\_insync.ticker.Ticker* method), 72  
 disconnect() (*ib\_insync.client.Client* method), 32  
 disconnect() (*ib\_insync.ib.IB* method), 7  
 DISCONNECTED (*ib\_insync.client.Client* attribute), 31  
 discretionaryAmt (*ib\_insync.order.Order* attribute), 38  
 discretionaryUpToLimitPrice (*ib\_insync.order.Order* attribute), 40  
 displayName (*ib\_insync.objects.SoftDollarTier* attribute), 76  
 displaySize (*ib\_insync.order.Order* attribute), 37  
 distance (*ib\_insync.contract.ScanData* attribute), 68  
 Dividends (class in *ib\_insync.objects*), 92  
 dividends (*ib\_insync.ticker.Ticker* attribute), 71  
 domAsks (*ib\_insync.ticker.Ticker* attribute), 71  
 domBids (*ib\_insync.ticker.Ticker* attribute), 71  
 DOMLevel (class in *ib\_insync.objects*), 90  
 domTicks (*ib\_insync.ticker.Ticker* attribute), 71  
 DoneStates (*ib\_insync.order.OrderStatus* attribute), 44  
 dontUseAutoPriceForHedge (*ib\_insync.order.Order* attribute), 40  
 download() (*ib\_insync.flexreport.FlexReport* method), 98  
 duration (*ib\_insync.objects.ConnectionStats* attribute), 93  
 duration (*ib\_insync.order.Order* attribute), 41  
 durationStr (*ib\_insync.objects.BarDataList* attribute), 93  
 DynamicObject (class in *ib\_insync.objects*), 94

## E

end (*ib\_insync.contract.TradingSession* attribute), 65  
 endDate (*ib\_insync.objects.WshEventData* attribute), 87  
 endDateTime (*ib\_insync.objects.BarDataList* attribute), 93  
 endDateTime (*ib\_insync.objects.HistoricalSchedule* attribute), 87  
 endDateTime (*ib\_insync.objects.HistoricalSession* attribute), 86  
 endTime (*ib\_insync.objects.RealTimeBar* attribute), 80  
 equityWithLoanAfter (*ib\_insync.order.OrderState* attribute), 45  
 equityWithLoanBefore (*ib\_insync.order.OrderState* attribute), 45

- `equityWithLoanChange` (*ib\_insync.order.OrderState attribute*), 45
  - `errorCode` (*ib\_insync.objects.TradeLogEntry attribute*), 85
  - `eTradeOnly` (*ib\_insync.order.Order attribute*), 38
  - `events` (*ib\_insync.client.Client attribute*), 30
  - `events` (*ib\_insync.ib.IB attribute*), 7
  - `events` (*ib\_insync.ibcontroller.Watchdog attribute*), 102
  - `events` (*ib\_insync.order.Trade attribute*), 47
  - `events` (*ib\_insync.ticker.Ticker attribute*), 69
  - `evMultiplier` (*ib\_insync.contract.ContractDetails attribute*), 66
  - `evMultiplier` (*ib\_insync.objects.Execution attribute*), 77
  - `evRule` (*ib\_insync.contract.ContractDetails attribute*), 66
  - `evRule` (*ib\_insync.objects.Execution attribute*), 77
  - `exch` (*ib\_insync.order.ExecutionCondition attribute*), 50
  - `exch` (*ib\_insync.order.PercentChangeCondition attribute*), 51
  - `exch` (*ib\_insync.order.PriceCondition attribute*), 48
  - `exch` (*ib\_insync.order.VolumeCondition attribute*), 51
  - `exchange` (*ib\_insync.contract.ComboLeg attribute*), 64
  - `exchange` (*ib\_insync.contract.Contract attribute*), 54
  - `exchange` (*ib\_insync.objects.DepthMktDataDescription attribute*), 84
  - `exchange` (*ib\_insync.objects.Execution attribute*), 77
  - `exchange` (*ib\_insync.objects.ExecutionFilter attribute*), 79
  - `exchange` (*ib\_insync.objects.HistoricalTickLast attribute*), 89
  - `exchange` (*ib\_insync.objects.OptionChain attribute*), 91
  - `exchange` (*ib\_insync.objects.SmartComponent attribute*), 93
  - `exchange` (*ib\_insync.objects.TickByTickAllLast attribute*), 89
  - `exchangeLetter` (*ib\_insync.objects.SmartComponent attribute*), 93
  - `excludeConvertible` (*ib\_insync.objects.ScannerSubscription attribute*), 76
  - `execId` (*ib\_insync.objects.CommissionReport attribute*), 78
  - `execId` (*ib\_insync.objects.Execution attribute*), 77
  - `Execution` (*class in ib\_insync.objects*), 77
  - `execution` (*ib\_insync.objects.Fill attribute*), 91
  - `ExecutionCondition` (*class in ib\_insync.order*), 50
  - `ExecutionFilter` (*class in ib\_insync.objects*), 79
  - `executions()` (*ib\_insync.ib.IB method*), 11
  - `exemptCode` (*ib\_insync.contract.ComboLeg attribute*), 64
  - `exemptCode` (*ib\_insync.order.Order attribute*), 38
  - `exerciseOptions()` (*ib\_insync.client.Client method*), 32
  - `exerciseOptions()` (*ib\_insync.ib.IB method*), 24
  - `expirations` (*ib\_insync.objects.OptionChain attribute*), 92
  - `extOperator` (*ib\_insync.order.Order attribute*), 40
  - `extract()` (*ib\_insync.flexreport.FlexReport method*), 98
  - `extraData` (*ib\_insync.objects.NewsTick attribute*), 92
- ## F
- `faGroup` (*ib\_insync.order.Order attribute*), 38
  - `faMethod` (*ib\_insync.order.Order attribute*), 38
  - `FamilyCode` (*class in ib\_insync.objects*), 92
  - `familyCodeStr` (*ib\_insync.objects.FamilyCode attribute*), 93
  - `faPercentage` (*ib\_insync.order.Order attribute*), 38
  - `faProfile` (*ib\_insync.order.Order attribute*), 38
  - `Fill` (*class in ib\_insync.objects*), 91
  - `fillCompetitors` (*ib\_insync.objects.WshEventData attribute*), 87
  - `Filled` (*ib\_insync.order.OrderStatus attribute*), 44
  - `filled` (*ib\_insync.order.OrderStatus attribute*), 43
  - `filled()` (*ib\_insync.order.Trade method*), 47
  - `filledQuantity` (*ib\_insync.order.Order attribute*), 41
  - `fillPortfolio` (*ib\_insync.objects.WshEventData attribute*), 87
  - `fills` (*ib\_insync.order.Trade attribute*), 46
  - `fills()` (*ib\_insync.ib.IB method*), 11
  - `fillWatchlist` (*ib\_insync.objects.WshEventData attribute*), 87
  - `filter` (*ib\_insync.objects.WshEventData attribute*), 87
  - `firmQuoteOnly` (*ib\_insync.order.Order attribute*), 38
  - `fixpassword` (*ib\_insync.ibcontroller.IBC attribute*), 100
  - `fixuserid` (*ib\_insync.ibcontroller.IBC attribute*), 100
  - `FlexError`, 98
  - `FlexReport` (*class in ib\_insync.flexreport*), 98
  - `Forex` (*class in ib\_insync.contract*), 58
  - `formatDate` (*ib\_insync.objects.BarDataList attribute*), 93
  - `formatIBDatetime()` (*in module ib\_insync.util*), 97
  - `formatSI()` (*in module ib\_insync.util*), 96
  - `FundamentalRatios` (*class in ib\_insync.objects*), 94
  - `fundamentalRatios` (*ib\_insync.ticker.Ticker attribute*), 71
  - `Future` (*class in ib\_insync.contract*), 56
  - `futuresOpenInterest` (*ib\_insync.ticker.Ticker attribute*), 71
  - `FuturesOption` (*class in ib\_insync.contract*), 61
- ## G
- `gamma` (*ib\_insync.objects.OptionComputation attribute*), 91
  - `gateway` (*ib\_insync.ibcontroller.IBC attribute*), 100
  - `getAccounts()` (*ib\_insync.client.Client method*), 31
  - `getLoop()` (*in module ib\_insync.util*), 97
  - `getReqId()` (*ib\_insync.client.Client method*), 31
  - `getWshEventData()` (*ib\_insync.ib.IB method*), 26

getWshEventDataAsync() (*ib\_insync.ib.IB* method), 29  
 getWshMetaData() (*ib\_insync.ib.IB* method), 26  
 getWshMetaDataAsync() (*ib\_insync.ib.IB* method), 29  
 globalErrorEvent (in module *ib\_insync.util*), 94  
 goodAfterTime (*ib\_insync.order.Order* attribute), 37  
 goodTillDate (*ib\_insync.order.Order* attribute), 37

## H

halted (*ib\_insync.ticker.Ticker* attribute), 70  
 hasBidAsk() (*ib\_insync.ticker.Ticker* method), 71  
 headline (*ib\_insync.objects.HistoricalNews* attribute), 92  
 headline (*ib\_insync.objects.NewsTick* attribute), 92  
 hedgeParam (*ib\_insync.order.Order* attribute), 39  
 hedgeType (*ib\_insync.order.Order* attribute), 39  
 hidden (*ib\_insync.order.Order* attribute), 37  
 high (*ib\_insync.objects.BarData* attribute), 79  
 high (*ib\_insync.objects.RealTimeBar* attribute), 80  
 high (*ib\_insync.ticker.Bar* attribute), 74  
 high (*ib\_insync.ticker.Ticker* attribute), 70  
 high13week (*ib\_insync.ticker.Ticker* attribute), 70  
 high26week (*ib\_insync.ticker.Ticker* attribute), 70  
 high52week (*ib\_insync.ticker.Ticker* attribute), 70  
 HistogramData (class in *ib\_insync.objects*), 82  
 HistoricalNews (class in *ib\_insync.objects*), 92  
 HistoricalSchedule (class in *ib\_insync.objects*), 87  
 HistoricalSession (class in *ib\_insync.objects*), 86  
 HistoricalTick (class in *ib\_insync.objects*), 88  
 HistoricalTickBidAsk (class in *ib\_insync.objects*), 88  
 HistoricalTickLast (class in *ib\_insync.objects*), 89  
 histVolatility (*ib\_insync.ticker.Ticker* attribute), 71  
 host (*ib\_insync.ibcontroller.Watchdog* attribute), 102

## I

IB (class in *ib\_insync.ib*), 5  
 ib (*ib\_insync.ibcontroller.Watchdog* attribute), 102  
 ib\_insync.client  
   module, 30  
 ib\_insync.contract  
   module, 52  
 ib\_insync.flexreport  
   module, 98  
 ib\_insync.ib  
   module, 5  
 ib\_insync.objects  
   module, 75  
 ib\_insync.order  
   module, 35  
 ib\_insync.ticker  
   module, 68  
 ib\_insync.util  
   module, 94  
 IBC (class in *ib\_insync.ibcontroller*), 98

ibcIni (*ib\_insync.ibcontroller.IBC* attribute), 100  
 IbcLogLevel (*ib\_insync.ibcontroller.IBC* attribute), 99  
 ibcPath (*ib\_insync.ibcontroller.IBC* attribute), 100  
 imbalanceOnly (*ib\_insync.order.Order* attribute), 41  
 impliedVol (*ib\_insync.objects.OptionComputation* attribute), 91  
 impliedVolatility (*ib\_insync.ticker.Ticker* attribute), 71  
 Inactive (*ib\_insync.order.OrderStatus* attribute), 44  
 includeExpired (*ib\_insync.contract.Contract* attribute), 54  
 increment (*ib\_insync.objects.PriceIncrement* attribute), 90  
 Index (class in *ib\_insync.contract*), 58  
 indexFuturePremium (*ib\_insync.ticker.Ticker* attribute), 71  
 industry (*ib\_insync.contract.ContractDetails* attribute), 65  
 initMarginAfter (*ib\_insync.order.OrderState* attribute), 45  
 initMarginBefore (*ib\_insync.order.OrderState* attribute), 45  
 initMarginChange (*ib\_insync.order.OrderState* attribute), 45  
 instrument (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 isActive() (*ib\_insync.order.Trade* method), 47  
 isConnected() (*ib\_insync.client.Client* method), 31  
 isConnected() (*ib\_insync.ib.IB* method), 7  
 isDone() (*ib\_insync.order.Trade* method), 47  
 isHashable() (*ib\_insync.contract.Contract* method), 54  
 isMore (*ib\_insync.order.MarginCondition* attribute), 49  
 isMore (*ib\_insync.order.PercentChangeCondition* attribute), 51  
 isMore (*ib\_insync.order.PriceCondition* attribute), 48  
 isMore (*ib\_insync.order.TimeCondition* attribute), 49  
 isMore (*ib\_insync.order.VolumeCondition* attribute), 51  
 isnamedtupleinstance() (in module *ib\_insync.util*), 95  
 isNaN() (in module *ib\_insync.util*), 95  
 isOmsContainer (*ib\_insync.order.Order* attribute), 40  
 isPeggedChangeAmountDecrease (*ib\_insync.order.Order* attribute), 40  
 isReady() (*ib\_insync.client.Client* method), 31  
 issueDate (*ib\_insync.contract.ContractDetails* attribute), 66  
 issuerId (*ib\_insync.contract.Contract* attribute), 54

## J

javaPath (*ib\_insync.ibcontroller.IBC* attribute), 100

## K

keepUpToDate (*ib\_insync.objects.BarDataList* attribute), 93

## L

last (*ib\_insync.ticker.Ticker* attribute), 69  
 lastExchange (*ib\_insync.ticker.Ticker* attribute), 70  
 lastFillPrice (*ib\_insync.order.OrderStatus* attribute), 44  
 lastGreeks (*ib\_insync.ticker.Ticker* attribute), 71  
 lastLiquidity (*ib\_insync.objects.Execution* attribute), 77  
 lastSize (*ib\_insync.ticker.Ticker* attribute), 69  
 lastTradeDateOrContractMonth (*ib\_insync.contract.Contract* attribute), 54  
 lastTradeTime (*ib\_insync.contract.ContractDetails* attribute), 66  
 lastYield (*ib\_insync.ticker.Ticker* attribute), 70  
 legsStr (*ib\_insync.contract.ScanData* attribute), 68  
 LimitOrder (class in *ib\_insync.order*), 41  
 liquidation (*ib\_insync.objects.Execution* attribute), 77  
 liquidHours (*ib\_insync.contract.ContractDetails* attribute), 66  
 liquidSessions() (*ib\_insync.contract.ContractDetails* method), 67  
 listingExch (*ib\_insync.objects.DepthMktDataDescription* attribute), 84  
 lmtPrice (*ib\_insync.order.Order* attribute), 37  
 lmtPriceOffset (*ib\_insync.order.Order* attribute), 40  
 load() (*ib\_insync.flexreport.FlexReport* method), 98  
 localSymbol (*ib\_insync.contract.Contract* attribute), 54  
 locationCode (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 log (*ib\_insync.order.Trade* attribute), 47  
 logToConsole() (in module *ib\_insync.util*), 95  
 logToFile() (in module *ib\_insync.util*), 95  
 longName (*ib\_insync.contract.ContractDetails* attribute), 65  
 loopUntil() (*ib\_insync.ib.IB* method), 9  
 low (*ib\_insync.objects.BarData* attribute), 79  
 low (*ib\_insync.objects.RealTimeBar* attribute), 80  
 low (*ib\_insync.ticker.Bar* attribute), 74  
 low (*ib\_insync.ticker.Ticker* attribute), 70  
 low13week (*ib\_insync.ticker.Ticker* attribute), 70  
 low26week (*ib\_insync.ticker.Ticker* attribute), 70  
 low52week (*ib\_insync.ticker.Ticker* attribute), 70  
 lowEdge (*ib\_insync.objects.PriceIncrement* attribute), 90

## M

maintMarginAfter (*ib\_insync.order.OrderState* attribute), 45  
 maintMarginBefore (*ib\_insync.order.OrderState* attribute), 45  
 maintMarginChange (*ib\_insync.order.OrderState* attribute), 45  
 managedAccounts() (*ib\_insync.ib.IB* method), 9  
 manualOrderTime (*ib\_insync.order.Order* attribute), 41  
 MarginCondition (class in *ib\_insync.order*), 49

marketCapAbove (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 marketCapBelow (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 marketDataType (*ib\_insync.ticker.Ticker* attribute), 69  
 marketMaker (*ib\_insync.objects.DOMLevel* attribute), 90  
 marketMaker (*ib\_insync.objects.MktDepthData* attribute), 90  
 marketName (*ib\_insync.contract.ContractDetails* attribute), 65  
 MarketOrder (class in *ib\_insync.order*), 42  
 marketPrice (*ib\_insync.objects.PortfolioItem* attribute), 90  
 marketPrice() (*ib\_insync.ticker.Ticker* method), 72  
 marketRuleIds (*ib\_insync.contract.ContractDetails* attribute), 66  
 marketValue (*ib\_insync.objects.PortfolioItem* attribute), 90  
 markPrice (*ib\_insync.ticker.Ticker* attribute), 70  
 maturity (*ib\_insync.contract.ContractDetails* attribute), 66  
 maturityDateAbove (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 maturityDateBelow (*ib\_insync.objects.ScannerSubscription* attribute), 76  
 MaxClientVersion (*ib\_insync.client.Client* attribute), 31  
 maxCommission (*ib\_insync.order.OrderState* attribute), 45  
 MaxRequests (*ib\_insync.client.Client* attribute), 30  
 MaxSyncedSubAccounts (*ib\_insync.ib.IB* attribute), 7  
 mdSizeMultiplier (*ib\_insync.contract.ContractDetails* attribute), 66  
 message (*ib\_insync.objects.NewsBulletin* attribute), 92  
 message (*ib\_insync.objects.TradeLogEntry* attribute), 85  
 midOffsetAtHalf (*ib\_insync.order.Order* attribute), 41  
 midOffsetAtWhole (*ib\_insync.order.Order* attribute), 41  
 midPoint (*ib\_insync.objects.TickByTickMidPoint* attribute), 90  
 midpoint() (*ib\_insync.ticker.Ticker* method), 72  
 Midpoints (class in *ib\_insync.ticker*), 73  
 midpoints() (*ib\_insync.ticker.TickerUpdateEvent* method), 73  
 mifid2DecisionAlgo (*ib\_insync.order.Order* attribute), 40  
 mifid2DecisionMaker (*ib\_insync.order.Order* attribute), 40  
 mifid2ExecutionAlgo (*ib\_insync.order.Order* attribute), 40  
 mifid2ExecutionTrader (*ib\_insync.order.Order* attribute), 40  
 MinClientVersion (*ib\_insync.client.Client* attribute),



30  
 minCommission (*ib\_insync.order.OrderState* attribute), 45  
 minCompeteSize (*ib\_insync.order.Order* attribute), 41  
 minQty (*ib\_insync.order.Order* attribute), 37  
 minSize (*ib\_insync.contract.ContractDetails* attribute), 66  
 minTick (*ib\_insync.contract.ContractDetails* attribute), 65  
 minTick (*ib\_insync.ticker.Ticker* attribute), 69  
 minTradeQty (*ib\_insync.order.Order* attribute), 41  
 mktCapPrice (*ib\_insync.order.OrderStatus* attribute), 44  
 MktDepthData (class in *ib\_insync.objects*), 90  
 modelCode (*ib\_insync.objects.AccountValue* attribute), 88  
 modelCode (*ib\_insync.objects.Execution* attribute), 77  
 modelCode (*ib\_insync.objects.PnL* attribute), 84  
 modelCode (*ib\_insync.objects.PnLSingle* attribute), 85  
 modelCode (*ib\_insync.order.Order* attribute), 40  
 modelGreeks (*ib\_insync.ticker.Ticker* attribute), 71  
 module  
     *ib\_insync.client*, 30  
     *ib\_insync.contract*, 52  
     *ib\_insync.flexreport*, 98  
     *ib\_insync.ib*, 5  
     *ib\_insync.objects*, 75  
     *ib\_insync.order*, 35  
     *ib\_insync.ticker*, 68  
     *ib\_insync.util*, 94  
 monitorAsync() (*ib\_insync.ibcontroller.IBC* method), 100  
 moodyRatingAbove (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 moodyRatingBelow (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 msgId (*ib\_insync.objects.NewsBulletin* attribute), 92  
 msgType (*ib\_insync.objects.NewsBulletin* attribute), 92  
 multiplier (*ib\_insync.contract.Contract* attribute), 54  
 multiplier (*ib\_insync.objects.OptionChain* attribute), 91  
 MutualFund (class in *ib\_insync.contract*), 61

## N

name (*ib\_insync.objects.NewsProvider* attribute), 83  
 name (*ib\_insync.objects.SoftDollarTier* attribute), 76  
 nbboPriceCap (*ib\_insync.order.Order* attribute), 38  
 NewsArticle (class in *ib\_insync.objects*), 92  
 NewsBulletin (class in *ib\_insync.objects*), 92  
 newsBulletins() (*ib\_insync.ib.IB* method), 12  
 NewsProvider (class in *ib\_insync.objects*), 83  
 NewsTick (class in *ib\_insync.objects*), 92  
 newsTicks() (*ib\_insync.ib.IB* method), 12  
 next12Months (*ib\_insync.objects.Dividends* attribute), 92  
 nextAmount (*ib\_insync.objects.Dividends* attribute), 92  
 nextDate (*ib\_insync.objects.Dividends* attribute), 92  
 nextOptionDate (*ib\_insync.contract.ContractDetails* attribute), 66  
 nextOptionPartial (*ib\_insync.contract.ContractDetails* attribute), 66  
 nextOptionType (*ib\_insync.contract.ContractDetails* attribute), 66  
 nonDefaults() (*ib\_insync.contract.Bag* method), 62  
 nonDefaults() (*ib\_insync.contract.Bond* method), 60  
 nonDefaults() (*ib\_insync.contract.CFD* method), 59  
 nonDefaults() (*ib\_insync.contract.ComboLeg* method), 64  
 nonDefaults() (*ib\_insync.contract.Commodity* method), 60  
 nonDefaults() (*ib\_insync.contract.ContFuture* method), 57  
 nonDefaults() (*ib\_insync.contract.Contract* method), 55  
 nonDefaults() (*ib\_insync.contract.ContractDescription* method), 67  
 nonDefaults() (*ib\_insync.contract.ContractDetails* method), 67  
 nonDefaults() (*ib\_insync.contract.Crypto* method), 63  
 nonDefaults() (*ib\_insync.contract.DeltaNeutralContract* method), 64  
 nonDefaults() (*ib\_insync.contract.Forex* method), 58  
 nonDefaults() (*ib\_insync.contract.Future* method), 57  
 nonDefaults() (*ib\_insync.contract.FuturesOption* method), 61  
 nonDefaults() (*ib\_insync.contract.Index* method), 59  
 nonDefaults() (*ib\_insync.contract.MutualFund* method), 61  
 nonDefaults() (*ib\_insync.contract.Option* method), 56  
 nonDefaults() (*ib\_insync.contract.ScanData* method), 68  
 nonDefaults() (*ib\_insync.contract.Stock* method), 55  
 nonDefaults() (*ib\_insync.contract.Warrant* method), 62  
 nonDefaults() (*ib\_insync.ibcontroller.IBC* method), 100  
 nonDefaults() (*ib\_insync.ibcontroller.Watchdog* method), 103  
 nonDefaults() (*ib\_insync.objects.BarData* method), 80  
 nonDefaults() (*ib\_insync.objects.CommissionReport* method), 78  
 nonDefaults() (*ib\_insync.objects.DepthMktDataDescription* method), 84  
 nonDefaults() (*ib\_insync.objects.Execution* method), 78  
 nonDefaults() (*ib\_insync.objects.ExecutionFilter* method), 79  
 nonDefaults() (*ib\_insync.objects.HistogramData* method), 83

- nonDefaults() (*ib\_insync.objects.HistoricalSchedule* method), 87  
 nonDefaults() (*ib\_insync.objects.HistoricalSession* method), 86  
 nonDefaults() (*ib\_insync.objects.NewsProvider* method), 83  
 nonDefaults() (*ib\_insync.objects.PnL* method), 84  
 nonDefaults() (*ib\_insync.objects.PnLSingle* method), 86  
 nonDefaults() (*ib\_insync.objects.RealTimeBar* method), 81  
 nonDefaults() (*ib\_insync.objects.ScannerSubscription* method), 76  
 nonDefaults() (*ib\_insync.objects.SoftDollarTier* method), 76  
 nonDefaults() (*ib\_insync.objects.TickAttrib* method), 81  
 nonDefaults() (*ib\_insync.objects.TickAttribBidAsk* method), 82  
 nonDefaults() (*ib\_insync.objects.TickAttribLast* method), 82  
 nonDefaults() (*ib\_insync.objects.TradeLogEntry* method), 85  
 nonDefaults() (*ib\_insync.objects.WshEventData* method), 88  
 nonDefaults() (*ib\_insync.order.ExecutionCondition* method), 50  
 nonDefaults() (*ib\_insync.order.LimitOrder* method), 42  
 nonDefaults() (*ib\_insync.order.MarginCondition* method), 50  
 nonDefaults() (*ib\_insync.order.MarketOrder* method), 42  
 nonDefaults() (*ib\_insync.order.Order* method), 41  
 nonDefaults() (*ib\_insync.order.OrderComboLeg* method), 46  
 nonDefaults() (*ib\_insync.order.OrderCondition* method), 48  
 nonDefaults() (*ib\_insync.order.OrderState* method), 45  
 nonDefaults() (*ib\_insync.order.OrderStatus* method), 44  
 nonDefaults() (*ib\_insync.order.PercentChangeCondition* method), 52  
 nonDefaults() (*ib\_insync.order.PriceCondition* method), 48  
 nonDefaults() (*ib\_insync.order.StopLimitOrder* method), 43  
 nonDefaults() (*ib\_insync.order.StopOrder* method), 42  
 nonDefaults() (*ib\_insync.order.TimeCondition* method), 49  
 nonDefaults() (*ib\_insync.order.Trade* method), 47  
 nonDefaults() (*ib\_insync.order.VolumeCondition* method), 51  
 nonDefaults() (*ib\_insync.ticker.Ticker* method), 72  
 notes (*ib\_insync.contract.ContractDetails* attribute), 66  
 notHeld (*ib\_insync.order.Order* attribute), 40  
 numberOfRows (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 numBytesRecv (*ib\_insync.objects.ConnectionStats* attribute), 93  
 numBytesSent (*ib\_insync.objects.ConnectionStats* attribute), 93  
 numMsgRecv (*ib\_insync.objects.ConnectionStats* attribute), 93  
 numMsgSent (*ib\_insync.objects.ConnectionStats* attribute), 93
- ## O
- ocaGroup (*ib\_insync.order.Order* attribute), 37  
 ocaType (*ib\_insync.order.Order* attribute), 37  
 on2fatimeout (*ib\_insync.ibcontroller.IBC* attribute), 100  
 on\_source() (*ib\_insync.ticker.Midpoints* method), 73  
 on\_source() (*ib\_insync.ticker.TickBars* method), 74  
 on\_source() (*ib\_insync.ticker.Tickfilter* method), 73  
 on\_source() (*ib\_insync.ticker.TimeBars* method), 74  
 on\_source() (*ib\_insync.ticker.VolumeBars* method), 75  
 oneCancelsAll() (*ib\_insync.ib.IB* static method), 13  
 open (*ib\_insync.objects.BarData* attribute), 79  
 open (*ib\_insync.ticker.Bar* attribute), 74  
 open (*ib\_insync.ticker.Ticker* attribute), 70  
 open\_ (*ib\_insync.objects.RealTimeBar* attribute), 80  
 openClose (*ib\_insync.contract.ComboLeg* attribute), 64  
 openClose (*ib\_insync.order.Order* attribute), 38  
 openOrders() (*ib\_insync.ib.IB* method), 11  
 openTrades() (*ib\_insync.ib.IB* method), 11  
 operation (*ib\_insync.objects.MktDepthData* attribute), 90  
 Option (class in *ib\_insync.contract*), 55  
 OptionChain (class in *ib\_insync.objects*), 91  
 OptionComputation (class in *ib\_insync.objects*), 91  
 optOutSmartRouting (*ib\_insync.order.Order* attribute), 38  
 optPrice (*ib\_insync.objects.OptionComputation* attribute), 91  
 Or() (*ib\_insync.order.OrderCondition* method), 48  
 Order (class in *ib\_insync.order*), 35  
 order (*ib\_insync.order.Trade* attribute), 46  
 OrderComboLeg (class in *ib\_insync.order*), 46  
 orderComboLegs (*ib\_insync.order.Order* attribute), 40  
 OrderCondition (class in *ib\_insync.order*), 48  
 orderId (*ib\_insync.objects.Execution* attribute), 77  
 orderId (*ib\_insync.order.Order* attribute), 37  
 orderId (*ib\_insync.order.OrderStatus* attribute), 43  
 orderMiscOptions (*ib\_insync.order.Order* attribute), 40  
 orderRef (*ib\_insync.objects.Execution* attribute), 77

orderRef (*ib\_insync.order.Order* attribute), 37  
 orders() (*ib\_insync.ib.IB* method), 11  
 OrderState (*class in ib\_insync.order*), 44  
 OrderStatus (*class in ib\_insync.order*), 43  
 orderStatus (*ib\_insync.order.Trade* attribute), 46  
 orderType (*ib\_insync.order.Order* attribute), 37  
 orderTypes (*ib\_insync.contract.ContractDetails* attribute), 65  
 origExchange (*ib\_insync.objects.NewsBulletin* attribute), 92  
 origin (*ib\_insync.order.Order* attribute), 38  
 outsideRth (*ib\_insync.order.Order* attribute), 37  
 overridePercentageConstraints (*ib\_insync.order.Order* attribute), 37

## P

pair() (*ib\_insync.contract.Forex* method), 58  
 parent (*ib\_insync.order.BasketOrder* attribute), 47  
 parentId (*ib\_insync.order.Order* attribute), 37  
 parentId (*ib\_insync.order.OrderStatus* attribute), 44  
 parentPermId (*ib\_insync.order.Order* attribute), 41  
 parseIBDatetime() (in module *ib\_insync.util*), 97  
 password (*ib\_insync.ibcontroller.IBC* attribute), 100  
 past12Months (*ib\_insync.objects.Dividends* attribute), 92  
 pastLimit (*ib\_insync.objects.TickAttrib* attribute), 81  
 pastLimit (*ib\_insync.objects.TickAttribLast* attribute), 82  
 patchAsyncio() (in module *ib\_insync.util*), 97  
 peggedChangeAmount (*ib\_insync.order.Order* attribute), 40  
 PendingCancel (*ib\_insync.order.OrderStatus* attribute), 44  
 pendingPriceRevision (*ib\_insync.objects.Execution* attribute), 77  
 PendingSubmit (*ib\_insync.order.OrderStatus* attribute), 44  
 pendingTickers() (*ib\_insync.ib.IB* method), 11  
 percent (*ib\_insync.order.MarginCondition* attribute), 50  
 PercentChangeCondition (*class in ib\_insync.order*), 51  
 percentOffset (*ib\_insync.order.Order* attribute), 37  
 permId (*ib\_insync.objects.Execution* attribute), 77  
 permId (*ib\_insync.order.Order* attribute), 37  
 permId (*ib\_insync.order.OrderStatus* attribute), 43  
 placeOrder() (*ib\_insync.client.Client* method), 32  
 placeOrder() (*ib\_insync.ib.IB* method), 13  
 PnL (*class in ib\_insync.objects*), 84  
 pnl() (*ib\_insync.ib.IB* method), 10  
 PnLSingle (*class in ib\_insync.objects*), 85  
 pnlSingle() (*ib\_insync.ib.IB* method), 10  
 port (*ib\_insync.ibcontroller.Watchdog* attribute), 102  
 portfolio() (*ib\_insync.ib.IB* method), 10  
 PortfolioItem (*class in ib\_insync.objects*), 90  
 Position (*class in ib\_insync.objects*), 91  
 position (*ib\_insync.objects.MktDepthData* attribute), 90  
 position (*ib\_insync.objects.PnLSingle* attribute), 86  
 position (*ib\_insync.objects.PortfolioItem* attribute), 90  
 position (*ib\_insync.objects.Position* attribute), 91  
 positions() (*ib\_insync.ib.IB* method), 10  
 postToAcs (*ib\_insync.order.Order* attribute), 41  
 preOpen (*ib\_insync.objects.TickAttrib* attribute), 81  
 PreSubmitted (*ib\_insync.order.OrderStatus* attribute), 44  
 prevAsk (*ib\_insync.ticker.Ticker* attribute), 70  
 prevAskSize (*ib\_insync.ticker.Ticker* attribute), 70  
 prevBid (*ib\_insync.ticker.Ticker* attribute), 70  
 prevBidSize (*ib\_insync.ticker.Ticker* attribute), 70  
 prevLast (*ib\_insync.ticker.Ticker* attribute), 70  
 prevLastSize (*ib\_insync.ticker.Ticker* attribute), 70  
 price (*ib\_insync.contract.DeltaNeutralContract* attribute), 64  
 price (*ib\_insync.objects.DOMLevel* attribute), 90  
 price (*ib\_insync.objects.Execution* attribute), 77  
 price (*ib\_insync.objects.HistogramData* attribute), 83  
 price (*ib\_insync.objects.HistoricalTick* attribute), 88  
 price (*ib\_insync.objects.HistoricalTickLast* attribute), 89  
 price (*ib\_insync.objects.MktDepthData* attribute), 90  
 price (*ib\_insync.objects.TickByTickAllLast* attribute), 89  
 price (*ib\_insync.objects.TickData* attribute), 88  
 price (*ib\_insync.order.OrderComboLeg* attribute), 46  
 price (*ib\_insync.order.PriceCondition* attribute), 48  
 priceAsk (*ib\_insync.objects.HistoricalTickBidAsk* attribute), 89  
 priceBid (*ib\_insync.objects.HistoricalTickBidAsk* attribute), 89  
 PriceCondition (*class in ib\_insync.order*), 48  
 PriceIncrement (*class in ib\_insync.objects*), 90  
 priceMagnifier (*ib\_insync.contract.ContractDetails* attribute), 65  
 primaryExchange (*ib\_insync.contract.Contract* attribute), 54  
 probeContract (*ib\_insync.ibcontroller.Watchdog* attribute), 102  
 probeTimeout (*ib\_insync.ibcontroller.Watchdog* attribute), 102  
 projection (*ib\_insync.contract.ScanData* attribute), 68  
 providerCode (*ib\_insync.objects.HistoricalNews* attribute), 92  
 providerCode (*ib\_insync.objects.NewsTick* attribute), 92  
 putable (*ib\_insync.contract.ContractDetails* attribute), 66  
 putOpenInterest (*ib\_insync.ticker.Ticker* attribute), 71  
 putVolume (*ib\_insync.ticker.Ticker* attribute), 71

pvDividend (*ib\_insync.objects.OptionComputation* attribute), 91

## Q

qualifyContracts() (*ib\_insync.ib.IB* method), 12

qualifyContractsAsync() (*ib\_insync.ib.IB* method), 27

queryDisplayGroups() (*ib\_insync.client.Client* method), 33

## R

RaiseRequestErrors (*ib\_insync.ib.IB* attribute), 7

raiseSyncErrors (*ib\_insync.ibcontroller.Watchdog* attribute), 102

randomizePrice (*ib\_insync.order.Order* attribute), 38

randomizeSize (*ib\_insync.order.Order* attribute), 38

rank (*ib\_insync.contract.ScanData* attribute), 68

ratings (*ib\_insync.contract.ContractDetails* attribute), 66

ratio (*ib\_insync.contract.ComboLeg* attribute), 63

readonly (*ib\_insync.ibcontroller.Watchdog* attribute), 102

realExpirationDate (*ib\_insync.contract.ContractDetails* attribute), 66

realizedPNL (*ib\_insync.objects.CommissionReport* attribute), 78

realizedPnL (*ib\_insync.objects.PnL* attribute), 84

realizedPnL (*ib\_insync.objects.PnLSingle* attribute), 86

realizedPNL (*ib\_insync.objects.PortfolioItem* attribute), 91

RealTimeBar (class in *ib\_insync.objects*), 80

RealTimeBarList (class in *ib\_insync.objects*), 93

realtimeBars() (*ib\_insync.ib.IB* method), 11

realTimeBarsOptions (*ib\_insync.objects.RealTimeBarList* attribute), 94

refDate (*ib\_insync.objects.HistoricalSession* attribute), 86

referenceChangeAmount (*ib\_insync.order.Order* attribute), 40

referenceContractId (*ib\_insync.order.Order* attribute), 40

referenceExchangeId (*ib\_insync.order.Order* attribute), 40

referencePriceType (*ib\_insync.order.Order* attribute), 39

refFuturesConId (*ib\_insync.order.Order* attribute), 41

regulatoryImbalance (*ib\_insync.ticker.Ticker* attribute), 71

remaining (*ib\_insync.order.OrderStatus* attribute), 43

remaining() (*ib\_insync.order.Trade* method), 47

replaceFA() (*ib\_insync.client.Client* method), 32

replaceFA() (*ib\_insync.ib.IB* method), 25

reqAccountSummary() (*ib\_insync.client.Client* method), 33

reqAccountSummary() (*ib\_insync.ib.IB* method), 14

reqAccountSummaryAsync() (*ib\_insync.ib.IB* method), 27

reqAccountUpdates() (*ib\_insync.client.Client* method), 32

reqAccountUpdates() (*ib\_insync.ib.IB* method), 13

reqAccountUpdatesAsync() (*ib\_insync.ib.IB* method), 27

reqAccountUpdatesMulti() (*ib\_insync.client.Client* method), 33

reqAccountUpdatesMulti() (*ib\_insync.ib.IB* method), 14

reqAccountUpdatesMultiAsync() (*ib\_insync.ib.IB* method), 27

reqAllOpenOrders() (*ib\_insync.client.Client* method), 32

reqAllOpenOrders() (*ib\_insync.ib.IB* method), 14

reqAllOpenOrdersAsync() (*ib\_insync.ib.IB* method), 27

reqAutoOpenOrders() (*ib\_insync.client.Client* method), 32

reqAutoOpenOrders() (*ib\_insync.ib.IB* method), 14

reqCompletedOrders() (*ib\_insync.client.Client* method), 34

reqCompletedOrders() (*ib\_insync.ib.IB* method), 14

reqCompletedOrdersAsync() (*ib\_insync.ib.IB* method), 27

reqContractDetails() (*ib\_insync.client.Client* method), 32

reqContractDetails() (*ib\_insync.ib.IB* method), 16

reqContractDetailsAsync() (*ib\_insync.ib.IB* method), 28

reqCurrentTime() (*ib\_insync.client.Client* method), 33

reqCurrentTime() (*ib\_insync.ib.IB* method), 13

reqCurrentTimeAsync() (*ib\_insync.ib.IB* method), 27

reqExecutions() (*ib\_insync.client.Client* method), 32

reqExecutions() (*ib\_insync.ib.IB* method), 15

reqExecutionsAsync() (*ib\_insync.ib.IB* method), 27

reqFamilyCodes() (*ib\_insync.client.Client* method), 34

reqFundamentalData() (*ib\_insync.client.Client* method), 33

reqFundamentalData() (*ib\_insync.ib.IB* method), 21

reqFundamentalDataAsync() (*ib\_insync.ib.IB* method), 28

reqGlobalCancel() (*ib\_insync.client.Client* method), 33

reqGlobalCancel() (*ib\_insync.ib.IB* method), 13

reqHeadTimeStamp() (*ib\_insync.client.Client* method), 34

reqHeadTimeStamp() (*ib\_insync.ib.IB* method), 19

reqHeadTimeStampAsync() (*ib\_insync.ib.IB* method), 28



reqHistogramData() (*ib\_insync.client.Client* method), 34  
 reqHistogramData() (*ib\_insync.ib.IB* method), 21  
 reqHistogramDataAsync() (*ib\_insync.ib.IB* method), 28  
 reqHistoricalData() (*ib\_insync.client.Client* method), 32  
 reqHistoricalData() (*ib\_insync.ib.IB* method), 17  
 reqHistoricalDataAsync() (*ib\_insync.ib.IB* method), 28  
 reqHistoricalNews() (*ib\_insync.client.Client* method), 34  
 reqHistoricalNews() (*ib\_insync.ib.IB* method), 24  
 reqHistoricalNewsAsync() (*ib\_insync.ib.IB* method), 29  
 reqHistoricalSchedule() (*ib\_insync.ib.IB* method), 18  
 reqHistoricalScheduleAsync() (*ib\_insync.ib.IB* method), 28  
 reqHistoricalTicks() (*ib\_insync.client.Client* method), 34  
 reqHistoricalTicks() (*ib\_insync.ib.IB* method), 18  
 reqHistoricalTicksAsync() (*ib\_insync.ib.IB* method), 28  
 reqId (*ib\_insync.objects.BarDataList* attribute), 93  
 reqId (*ib\_insync.objects.RealTimeBarList* attribute), 94  
 reqId (*ib\_insync.objects.ScanDataList* attribute), 94  
 reqIds() (*ib\_insync.client.Client* method), 32  
 reqManagedAccts() (*ib\_insync.client.Client* method), 32  
 reqMarketDataType() (*ib\_insync.client.Client* method), 33  
 reqMarketDataType() (*ib\_insync.ib.IB* method), 19  
 reqMarketRule() (*ib\_insync.client.Client* method), 34  
 reqMarketRule() (*ib\_insync.ib.IB* method), 16  
 reqMarketRuleAsync() (*ib\_insync.ib.IB* method), 28  
 reqMatchingSymbols() (*ib\_insync.client.Client* method), 34  
 reqMatchingSymbols() (*ib\_insync.ib.IB* method), 16  
 reqMatchingSymbolsAsync() (*ib\_insync.ib.IB* method), 28  
 reqMktData() (*ib\_insync.client.Client* method), 32  
 reqMktData() (*ib\_insync.ib.IB* method), 19  
 reqMktDepth() (*ib\_insync.client.Client* method), 32  
 reqMktDepth() (*ib\_insync.ib.IB* method), 21  
 reqMktDepthExchanges() (*ib\_insync.client.Client* method), 34  
 reqMktDepthExchanges() (*ib\_insync.ib.IB* method), 21  
 reqMktDepthExchangesAsync() (*ib\_insync.ib.IB* method), 28  
 reqNewsArticle() (*ib\_insync.client.Client* method), 34  
 reqNewsArticle() (*ib\_insync.ib.IB* method), 24  
 reqNewsArticleAsync() (*ib\_insync.ib.IB* method), 29  
 reqNewsBulletins() (*ib\_insync.client.Client* method), 32  
 reqNewsBulletins() (*ib\_insync.ib.IB* method), 25  
 reqNewsProviders() (*ib\_insync.client.Client* method), 34  
 reqNewsProviders() (*ib\_insync.ib.IB* method), 24  
 reqNewsProvidersAsync() (*ib\_insync.ib.IB* method), 29  
 reqOpenOrders() (*ib\_insync.client.Client* method), 32  
 reqOpenOrders() (*ib\_insync.ib.IB* method), 14  
 reqOpenOrdersAsync() (*ib\_insync.ib.IB* method), 27  
 reqPnL() (*ib\_insync.client.Client* method), 34  
 reqPnL() (*ib\_insync.ib.IB* method), 15  
 reqPnLSingle() (*ib\_insync.client.Client* method), 34  
 reqPnLSingle() (*ib\_insync.ib.IB* method), 15  
 reqPositions() (*ib\_insync.client.Client* method), 33  
 reqPositions() (*ib\_insync.ib.IB* method), 15  
 reqPositionsAsync() (*ib\_insync.ib.IB* method), 28  
 reqPositionsMulti() (*ib\_insync.client.Client* method), 33  
 reqRealTimeBars() (*ib\_insync.client.Client* method), 33  
 reqRealTimeBars() (*ib\_insync.ib.IB* method), 16  
 reqScannerData() (*ib\_insync.ib.IB* method), 22  
 reqScannerDataAsync() (*ib\_insync.ib.IB* method), 29  
 reqScannerParameters() (*ib\_insync.client.Client* method), 33  
 reqScannerParameters() (*ib\_insync.ib.IB* method), 23  
 reqScannerParametersAsync() (*ib\_insync.ib.IB* method), 29  
 reqScannerSubscription() (*ib\_insync.client.Client* method), 32  
 reqScannerSubscription() (*ib\_insync.ib.IB* method), 22  
 reqSecDefOptParams() (*ib\_insync.client.Client* method), 34  
 reqSecDefOptParams() (*ib\_insync.ib.IB* method), 23  
 reqSecDefOptParamsAsync() (*ib\_insync.ib.IB* method), 29  
 reqSmartComponents() (*ib\_insync.client.Client* method), 34  
 reqSmartComponents() (*ib\_insync.ib.IB* method), 21  
 reqSmartComponentsAsync() (*ib\_insync.ib.IB* method), 28  
 reqSoftDollarTiers() (*ib\_insync.client.Client* method), 34  
 reqTickByTickData() (*ib\_insync.client.Client* method), 34  
 reqTickByTickData() (*ib\_insync.ib.IB* method), 20  
 reqTickers() (*ib\_insync.ib.IB* method), 12  
 reqTickersAsync() (*ib\_insync.ib.IB* method), 27  
 RequestError (class in *ib\_insync.wrapper*), 94  
 requestFA() (*ib\_insync.client.Client* method), 32

requestFA() (*ib\_insync.ib.IB* method), 25  
 requestFAAsync() (*ib\_insync.ib.IB* method), 29  
 RequestsInterval (*ib\_insync.client.Client* attribute), 30  
 RequestTimeout (*ib\_insync.ib.IB* attribute), 7  
 reqUserInfo() (*ib\_insync.client.Client* method), 34  
 reqUserInfo() (*ib\_insync.ib.IB* method), 26  
 reqUserInfoAsync() (*ib\_insync.ib.IB* method), 29  
 reqWshEventData() (*ib\_insync.client.Client* method), 34  
 reqWshEventData() (*ib\_insync.ib.IB* method), 26  
 reqWshMetaData() (*ib\_insync.client.Client* method), 34  
 reqWshMetaData() (*ib\_insync.ib.IB* method), 25  
 reset() (*ib\_insync.client.Client* method), 31  
 retryDelay (*ib\_insync.ibcontroller.Watchdog* attribute), 102  
 right (*ib\_insync.contract.Contract* attribute), 54  
 root (*ib\_insync.flexreport.FlexReport* attribute), 98  
 routeMarketableToBbo (*ib\_insync.order.Order* attribute), 41  
 rtHistVolatility (*ib\_insync.ticker.Ticker* attribute), 70  
 rtTime (*ib\_insync.ticker.Ticker* attribute), 70  
 rtTradeVolume (*ib\_insync.ticker.Ticker* attribute), 70  
 rtVolume (*ib\_insync.ticker.Ticker* attribute), 70  
 rule80A (*ib\_insync.order.Order* attribute), 37  
 run() (*ib\_insync.client.Client* method), 31  
 run() (*ib\_insync.ib.IB* static method), 8  
 run() (in module *ib\_insync.util*), 96  
 runAsync() (*ib\_insync.ibcontroller.Watchdog* method), 102

## S

save() (*ib\_insync.flexreport.FlexReport* method), 98  
 scaleAutoReset (*ib\_insync.order.Order* attribute), 39  
 scaleInitFillQty (*ib\_insync.order.Order* attribute), 39  
 scaleInitLevelSize (*ib\_insync.order.Order* attribute), 39  
 scaleInitPosition (*ib\_insync.order.Order* attribute), 39  
 scalePriceAdjustInterval (*ib\_insync.order.Order* attribute), 39  
 scalePriceAdjustValue (*ib\_insync.order.Order* attribute), 39  
 scalePriceIncrement (*ib\_insync.order.Order* attribute), 39  
 scaleProfitOffset (*ib\_insync.order.Order* attribute), 39  
 scaleRandomPercent (*ib\_insync.order.Order* attribute), 39  
 scaleSubsLevelSize (*ib\_insync.order.Order* attribute), 39  
 scaleTable (*ib\_insync.order.Order* attribute), 39  
 scanCode (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 ScanData (class in *ib\_insync.contract*), 68  
 ScanDataList (class in *ib\_insync.objects*), 94  
 scannerSettingPairs (*ib\_insync.objects.ScannerSubscription* attribute), 76  
 ScannerSubscription (class in *ib\_insync.objects*), 75  
 scannerSubscriptionFilterOptions (*ib\_insync.objects.ScanDataList* attribute), 94  
 scannerSubscriptionOptions (*ib\_insync.objects.ScanDataList* attribute), 94  
 schedule() (*ib\_insync.ib.IB* static method), 8  
 schedule() (in module *ib\_insync.util*), 96  
 secId (*ib\_insync.contract.Contract* attribute), 54  
 secIdList (*ib\_insync.contract.ContractDetails* attribute), 66  
 secIdType (*ib\_insync.contract.Contract* attribute), 54  
 secType (*ib\_insync.contract.Contract* attribute), 54  
 secType (*ib\_insync.objects.DepthMktDataDescription* attribute), 84  
 secType (*ib\_insync.objects.ExecutionFilter* attribute), 79  
 secType (*ib\_insync.order.ExecutionCondition* attribute), 50  
 send() (*ib\_insync.client.Client* method), 32  
 sendMsg() (*ib\_insync.client.Client* method), 32  
 serverVersion() (*ib\_insync.client.Client* method), 31  
 serviceDataType (*ib\_insync.objects.DepthMktDataDescription* attribute), 84  
 sessions (*ib\_insync.objects.HistoricalSchedule* attribute), 87  
 setConnectOptions() (*ib\_insync.client.Client* method), 31  
 setServerLogLevel() (*ib\_insync.client.Client* method), 32  
 setTimeout() (*ib\_insync.ib.IB* method), 9  
 settlingFirm (*ib\_insync.order.Order* attribute), 39  
 shareholder (*ib\_insync.order.Order* attribute), 41  
 shares (*ib\_insync.objects.Execution* attribute), 77  
 shortableShares (*ib\_insync.ticker.Ticker* attribute), 71  
 shortSaleSlot (*ib\_insync.contract.ComboLeg* attribute), 64  
 shortSaleSlot (*ib\_insync.order.Order* attribute), 38  
 side (*ib\_insync.objects.Execution* attribute), 77  
 side (*ib\_insync.objects.ExecutionFilter* attribute), 79  
 side (*ib\_insync.objects.MktDepthData* attribute), 90  
 size (*ib\_insync.objects.DOMLevel* attribute), 90  
 size (*ib\_insync.objects.HistoricalTick* attribute), 88  
 size (*ib\_insync.objects.HistoricalTickLast* attribute), 89  
 size (*ib\_insync.objects.MktDepthData* attribute), 90  
 size (*ib\_insync.objects.TickByTickAllLast* attribute), 89  
 size (*ib\_insync.objects.TickData* attribute), 88

sizeAsk (*ib\_insync.objects.HistoricalTickBidAsk* attribute), 89  
 sizeBid (*ib\_insync.objects.HistoricalTickBidAsk* attribute), 89  
 sizeIncrement (*ib\_insync.contract.ContractDetails* attribute), 66  
 sleep() (*ib\_insync.ib.IB* static method), 8  
 sleep() (in module *ib\_insync.util*), 96  
 smartComboRoutingParams (*ib\_insync.order.Order* attribute), 39  
 SmartComponent (class in *ib\_insync.objects*), 93  
 snapshotPermissions (*ib\_insync.ticker.Ticker* attribute), 71  
 SoftDollarTier (class in *ib\_insync.objects*), 76  
 softDollarTier (*ib\_insync.order.Order* attribute), 40  
 solicited (*ib\_insync.order.Order* attribute), 40  
 specialConditions (*ib\_insync.objects.HistoricalTickLast* attribute), 89  
 specialConditions (*ib\_insync.objects.TickByTickAllLast* attribute), 89  
 spRatingAbove (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 spRatingBelow (*ib\_insync.objects.ScannerSubscription* attribute), 75  
 start (*ib\_insync.contract.TradingSession* attribute), 65  
 start() (*ib\_insync.ibcontroller.IBC* method), 100  
 start() (*ib\_insync.ibcontroller.Watchdog* method), 102  
 startApi() (*ib\_insync.client.Client* method), 33  
 startAsync() (*ib\_insync.ibcontroller.IBC* method), 100  
 startDate (*ib\_insync.objects.WshEventData* attribute), 87  
 startDateTime (*ib\_insync.objects.HistoricalSchedule* attribute), 87  
 startDateTime (*ib\_insync.objects.HistoricalSession* attribute), 86  
 startingPrice (*ib\_insync.order.Order* attribute), 38  
 startLoop() (in module *ib\_insync.util*), 97  
 startTime (*ib\_insync.objects.ConnectionStats* attribute), 93  
 status (*ib\_insync.objects.TradeLogEntry* attribute), 85  
 status (*ib\_insync.order.OrderState* attribute), 45  
 status (*ib\_insync.order.OrderStatus* attribute), 43  
 Stock (class in *ib\_insync.contract*), 55  
 stockRangeLower (*ib\_insync.order.Order* attribute), 38  
 stockRangeUpper (*ib\_insync.order.Order* attribute), 38  
 stockRefPrice (*ib\_insync.order.Order* attribute), 38  
 stockType (*ib\_insync.contract.ContractDetails* attribute), 66  
 stockTypeFilter (*ib\_insync.objects.ScannerSubscription* attribute), 76  
 stop() (*ib\_insync.ibcontroller.Watchdog* method), 102  
 StopLimitOrder (class in *ib\_insync.order*), 43  
 stopLoss (*ib\_insync.order.BasketOrder* attribute), 48  
 StopOrder (class in *ib\_insync.order*), 42  
 strike (*ib\_insync.contract.Contract* attribute), 54  
 strikes (*ib\_insync.objects.OptionChain* attribute), 92  
 subcategory (*ib\_insync.contract.ContractDetails* attribute), 65  
 Submitted (*ib\_insync.order.OrderStatus* attribute), 44  
 subscribeToGroupEvents() (*ib\_insync.client.Client* method), 33  
 subscription (*ib\_insync.objects.ScanDataList* attribute), 94  
 suggestedSizeIncrement (*ib\_insync.contract.ContractDetails* attribute), 66  
 sweepToFill (*ib\_insync.order.Order* attribute), 37  
 symbol (*ib\_insync.contract.Contract* attribute), 54  
 symbol (*ib\_insync.objects.ExecutionFilter* attribute), 79  
 symbol (*ib\_insync.order.ExecutionCondition* attribute), 50

## T

tag (*ib\_insync.contract.TagValue* attribute), 63  
 tag (*ib\_insync.objects.AccountValue* attribute), 88  
 TagValue (class in *ib\_insync.contract*), 63  
 takeProfit (*ib\_insync.order.BasketOrder* attribute), 48  
 terminate() (*ib\_insync.ibcontroller.IBC* method), 100  
 terminateAsync() (*ib\_insync.ibcontroller.IBC* method), 100  
 theta (*ib\_insync.objects.OptionComputation* attribute), 91  
 TickAttrib (class in *ib\_insync.objects*), 81  
 tickAttrib (*ib\_insync.objects.OptionComputation* attribute), 91  
 TickAttribBidAsk (class in *ib\_insync.objects*), 81  
 tickAttribBidAsk (*ib\_insync.objects.HistoricalTickBidAsk* attribute), 89  
 tickAttribBidAsk (*ib\_insync.objects.TickByTickBidAsk* attribute), 90  
 TickAttribLast (class in *ib\_insync.objects*), 82  
 tickAttribLast (*ib\_insync.objects.HistoricalTickLast* attribute), 89  
 tickAttribLast (*ib\_insync.objects.TickByTickAllLast* attribute), 89  
 TickBars (class in *ib\_insync.ticker*), 74  
 tickbars() (*ib\_insync.ticker.Tickfilter* method), 73  
 TickByTickAllLast (class in *ib\_insync.objects*), 89  
 TickByTickBidAsk (class in *ib\_insync.objects*), 89  
 TickByTickMidPoint (class in *ib\_insync.objects*), 90  
 tickByTicks (*ib\_insync.ticker.Ticker* attribute), 71  
 TickData (class in *ib\_insync.objects*), 88  
 Ticker (class in *ib\_insync.ticker*), 68  
 ticker() (*ib\_insync.ib.IB* method), 11  
 tickers() (*ib\_insync.ib.IB* method), 11  
 TickerUpdateEvent (class in *ib\_insync.ticker*), 72  
 Tickfilter (class in *ib\_insync.ticker*), 73

ticks (*ib\_insync.ticker.Ticker* attribute), 71  
 tickType (*ib\_insync.objects.TickByTickAllLast* attribute), 89  
 tickType (*ib\_insync.objects.TickData* attribute), 88  
 tif (*ib\_insync.order.Order* attribute), 37  
 time (*ib\_insync.objects.Execution* attribute), 77  
 time (*ib\_insync.objects.ExecutionFilter* attribute), 79  
 time (*ib\_insync.objects.Fill* attribute), 91  
 time (*ib\_insync.objects.HistoricalNews* attribute), 92  
 time (*ib\_insync.objects.HistoricalTick* attribute), 88  
 time (*ib\_insync.objects.HistoricalTickBidAsk* attribute), 89  
 time (*ib\_insync.objects.HistoricalTickLast* attribute), 89  
 time (*ib\_insync.objects.MktDepthData* attribute), 90  
 time (*ib\_insync.objects.RealTimeBar* attribute), 80  
 time (*ib\_insync.objects.TickByTickAllLast* attribute), 89  
 time (*ib\_insync.objects.TickByTickBidAsk* attribute), 89  
 time (*ib\_insync.objects.TickByTickMidPoint* attribute), 90  
 time (*ib\_insync.objects.TickData* attribute), 88  
 time (*ib\_insync.objects.TradeLogEntry* attribute), 85  
 time (*ib\_insync.order.TimeCondition* attribute), 49  
 time (*ib\_insync.ticker.Bar* attribute), 74  
 time (*ib\_insync.ticker.Ticker* attribute), 69  
 TimeBars (*class in ib\_insync.ticker*), 74  
 timebars() (*ib\_insync.ticker.Tickfilter* method), 73  
 TimeCondition (*class in ib\_insync.order*), 49  
 timeit (*class in ib\_insync.util*), 96  
 timeRange() (*ib\_insync.ib.IB* static method), 8  
 timeRange() (*in module ib\_insync.util*), 96  
 timeRangeAsync() (*ib\_insync.ib.IB* static method), 8  
 timeRangeAsync() (*in module ib\_insync.util*), 97  
 timeStamp (*ib\_insync.objects.NewsTick* attribute), 92  
 timeZone (*ib\_insync.objects.HistoricalSchedule* attribute), 87  
 timeZoneId (*ib\_insync.contract.ContractDetails* attribute), 65  
 TimezoneTWS (*ib\_insync.ib.IB* attribute), 7  
 topics() (*ib\_insync.flexreport.FlexReport* method), 98  
 totalLimit (*ib\_insync.objects.WshEventData* attribute), 87  
 totalQuantity (*ib\_insync.order.Order* attribute), 37  
 Trade (*class in ib\_insync.order*), 46  
 tradeCount (*ib\_insync.ticker.Ticker* attribute), 70  
 TradeLogEntry (*class in ib\_insync.objects*), 85  
 tradeRate (*ib\_insync.ticker.Ticker* attribute), 70  
 trades() (*ib\_insync.ib.IB* method), 10  
 trades() (*ib\_insync.ticker.TickerUpdateEvent* method), 72  
 tradingClass (*ib\_insync.contract.Contract* attribute), 54  
 tradingClass (*ib\_insync.objects.OptionChain* attribute), 91  
 tradingHours (*ib\_insync.contract.ContractDetails* attribute), 66  
 tradingMode (*ib\_insync.ibcontroller.IBC* attribute), 100  
 TradingSession (*class in ib\_insync.contract*), 65  
 tradingSessions() (*ib\_insync.contract.ContractDetails* method), 67  
 trailingPercent (*ib\_insync.order.Order* attribute), 38  
 trailStopPrice (*ib\_insync.order.Order* attribute), 38  
 transmit (*ib\_insync.order.Order* attribute), 37  
 tree() (*in module ib\_insync.util*), 95  
 triggerMethod (*ib\_insync.order.Order* attribute), 37  
 triggerMethod (*ib\_insync.order.PriceCondition* attribute), 48  
 triggerPrice (*ib\_insync.order.Order* attribute), 40  
 tuple() (*ib\_insync.contract.Bag* method), 62  
 tuple() (*ib\_insync.contract.Bond* method), 60  
 tuple() (*ib\_insync.contract.CFD* method), 59  
 tuple() (*ib\_insync.contract.ComboLeg* method), 64  
 tuple() (*ib\_insync.contract.Commodity* method), 60  
 tuple() (*ib\_insync.contract.ContFuture* method), 57  
 tuple() (*ib\_insync.contract.Contract* method), 55  
 tuple() (*ib\_insync.contract.ContractDescription* method), 67  
 tuple() (*ib\_insync.contract.ContractDetails* method), 67  
 tuple() (*ib\_insync.contract.Crypto* method), 63  
 tuple() (*ib\_insync.contract.DeltaNeutralContract* method), 64  
 tuple() (*ib\_insync.contract.Forex* method), 58  
 tuple() (*ib\_insync.contract.Future* method), 57  
 tuple() (*ib\_insync.contract.FuturesOption* method), 61  
 tuple() (*ib\_insync.contract.Index* method), 59  
 tuple() (*ib\_insync.contract.MutualFund* method), 62  
 tuple() (*ib\_insync.contract.Option* method), 56  
 tuple() (*ib\_insync.contract.ScanData* method), 68  
 tuple() (*ib\_insync.contract.Stock* method), 55  
 tuple() (*ib\_insync.contract.Warrant* method), 62  
 tuple() (*ib\_insync.ibcontroller.IBC* method), 100  
 tuple() (*ib\_insync.ibcontroller.Watchdog* method), 103  
 tuple() (*ib\_insync.objects.BarData* method), 80  
 tuple() (*ib\_insync.objects.CommissionReport* method), 78  
 tuple() (*ib\_insync.objects.DepthMktDataDescription* method), 84  
 tuple() (*ib\_insync.objects.Execution* method), 78  
 tuple() (*ib\_insync.objects.ExecutionFilter* method), 79  
 tuple() (*ib\_insync.objects.HistogramData* method), 83  
 tuple() (*ib\_insync.objects.HistoricalSchedule* method), 87  
 tuple() (*ib\_insync.objects.HistoricalSession* method), 86  
 tuple() (*ib\_insync.objects.NewsProvider* method), 83  
 tuple() (*ib\_insync.objects.PnL* method), 85  
 tuple() (*ib\_insync.objects.PnLSingle* method), 86



tuple() (*ib\_insync.objects.RealTimeBar* method), 81  
 tuple() (*ib\_insync.objects.ScannerSubscription* method), 76  
 tuple() (*ib\_insync.objects.SoftDollarTier* method), 76  
 tuple() (*ib\_insync.objects.TickAttrib* method), 81  
 tuple() (*ib\_insync.objects.TickAttribBidAsk* method), 82  
 tuple() (*ib\_insync.objects.TickAttribLast* method), 82  
 tuple() (*ib\_insync.objects.TradeLogEntry* method), 85  
 tuple() (*ib\_insync.objects.WshEventData* method), 88  
 tuple() (*ib\_insync.order.ExecutionCondition* method), 50  
 tuple() (*ib\_insync.order.LimitOrder* method), 42  
 tuple() (*ib\_insync.order.MarginCondition* method), 50  
 tuple() (*ib\_insync.order.MarketOrder* method), 42  
 tuple() (*ib\_insync.order.Order* method), 41  
 tuple() (*ib\_insync.order.OrderComboLeg* method), 46  
 tuple() (*ib\_insync.order.OrderCondition* method), 48  
 tuple() (*ib\_insync.order.OrderState* method), 45  
 tuple() (*ib\_insync.order.OrderStatus* method), 44  
 tuple() (*ib\_insync.order.PercentChangeCondition* method), 52  
 tuple() (*ib\_insync.order.PriceCondition* method), 49  
 tuple() (*ib\_insync.order.StopLimitOrder* method), 43  
 tuple() (*ib\_insync.order.StopOrder* method), 43  
 tuple() (*ib\_insync.order.TimeCondition* method), 49  
 tuple() (*ib\_insync.order.Trade* method), 47  
 tuple() (*ib\_insync.order.VolumeCondition* method), 51  
 tuple() (*ib\_insync.ticker.Ticker* method), 72  
 twsPath (*ib\_insync.ibcontroller.IBC* attribute), 100  
 twsSettingsPath (*ib\_insync.ibcontroller.IBC* attribute), 100  
 twsVersion (*ib\_insync.ibcontroller.IBC* attribute), 100

## U

underConId (*ib\_insync.contract.ContractDetails* attribute), 65  
 underlyingConId (*ib\_insync.objects.OptionChain* attribute), 91  
 underSecType (*ib\_insync.contract.ContractDetails* attribute), 66  
 underSymbol (*ib\_insync.contract.ContractDetails* attribute), 66  
 undPrice (*ib\_insync.objects.OptionComputation* attribute), 91  
 unrealizedPnL (*ib\_insync.objects.PnL* attribute), 84  
 unrealizedPnL (*ib\_insync.objects.PnLSingle* attribute), 86  
 unrealizedPNL (*ib\_insync.objects.PortfolioItem* attribute), 90  
 unreported (*ib\_insync.objects.TickAttribLast* attribute), 82  
 unsubscribeFromGroupEvents()  
     (*ib\_insync.client.Client* method), 33  
 update() (*ib\_insync.contract.Bag* method), 63  
 update() (*ib\_insync.contract.Bond* method), 60  
 update() (*ib\_insync.contract.CFD* method), 59  
 update() (*ib\_insync.contract.ComboLeg* method), 64  
 update() (*ib\_insync.contract.Commodity* method), 60  
 update() (*ib\_insync.contract.ContFuture* method), 58  
 update() (*ib\_insync.contract.Contract* method), 55  
 update() (*ib\_insync.contract.ContractDescription* method), 67  
 update() (*ib\_insync.contract.ContractDetails* method), 67  
 update() (*ib\_insync.contract.Crypto* method), 63  
 update() (*ib\_insync.contract.DeltaNeutralContract* method), 65  
 update() (*ib\_insync.contract.Forex* method), 58  
 update() (*ib\_insync.contract.Future* method), 57  
 update() (*ib\_insync.contract.FuturesOption* method), 61  
 update() (*ib\_insync.contract.Index* method), 59  
 update() (*ib\_insync.contract.MutualFund* method), 62  
 update() (*ib\_insync.contract.Option* method), 56  
 update() (*ib\_insync.contract.ScanData* method), 68  
 update() (*ib\_insync.contract.Stock* method), 55  
 update() (*ib\_insync.contract.Warrant* method), 62  
 update() (*ib\_insync.ibcontroller.IBC* method), 100  
 update() (*ib\_insync.ibcontroller.Watchdog* method), 103  
 update() (*ib\_insync.objects.BarData* method), 80  
 update() (*ib\_insync.objects.CommissionReport* method), 78  
 update() (*ib\_insync.objects.DepthMktDataDescription* method), 84  
 update() (*ib\_insync.objects.Execution* method), 78  
 update() (*ib\_insync.objects.ExecutionFilter* method), 79  
 update() (*ib\_insync.objects.HistogramData* method), 83  
 update() (*ib\_insync.objects.HistoricalSchedule* method), 87  
 update() (*ib\_insync.objects.HistoricalSession* method), 86  
 update() (*ib\_insync.objects.NewsProvider* method), 83  
 update() (*ib\_insync.objects.PnL* method), 85  
 update() (*ib\_insync.objects.PnLSingle* method), 86  
 update() (*ib\_insync.objects.RealTimeBar* method), 81  
 update() (*ib\_insync.objects.ScannerSubscription* method), 76  
 update() (*ib\_insync.objects.SoftDollarTier* method), 77  
 update() (*ib\_insync.objects.TickAttrib* method), 81  
 update() (*ib\_insync.objects.TickAttribBidAsk* method), 82  
 update() (*ib\_insync.objects.TickAttribLast* method), 82  
 update() (*ib\_insync.objects.TradeLogEntry* method), 85  
 update() (*ib\_insync.objects.WshEventData* method), 88

`update()` (*ib\_insync.order.ExecutionCondition* method), 50  
`update()` (*ib\_insync.order.LimitOrder* method), 42  
`update()` (*ib\_insync.order.MarginCondition* method), 50  
`update()` (*ib\_insync.order.MarketOrder* method), 42  
`update()` (*ib\_insync.order.Order* method), 41  
`update()` (*ib\_insync.order.OrderComboLeg* method), 46  
`update()` (*ib\_insync.order.OrderCondition* method), 48  
`update()` (*ib\_insync.order.OrderState* method), 45  
`update()` (*ib\_insync.order.OrderStatus* method), 44  
`update()` (*ib\_insync.order.PercentChangeCondition* method), 52  
`update()` (*ib\_insync.order.PriceCondition* method), 49  
`update()` (*ib\_insync.order.StopLimitOrder* method), 43  
`update()` (*ib\_insync.order.StopOrder* method), 43  
`update()` (*ib\_insync.order.TimeCondition* method), 49  
`update()` (*ib\_insync.order.Trade* method), 47  
`update()` (*ib\_insync.order.VolumeCondition* method), 51  
`update()` (*ib\_insync.ticker.Ticker* method), 72  
`updateDisplayGroup()` (*ib\_insync.client.Client* method), 33  
`updateReqId()` (*ib\_insync.client.Client* method), 31  
`usePriceMgmtAlgo` (*ib\_insync.order.Order* attribute), 41  
`useQt()` (in module *ib\_insync.util*), 97  
`userid` (*ib\_insync.ibcontroller.IBC* attribute), 100  
`useRTH` (*ib\_insync.objects.BarDataList* attribute), 93  
`useRTH` (*ib\_insync.objects.RealTimeBarList* attribute), 94

## V

`val` (*ib\_insync.objects.SoftDollarTier* attribute), 76  
`validExchanges` (*ib\_insync.contract.ContractDetails* attribute), 65  
`value` (*ib\_insync.contract.TagValue* attribute), 63  
`value` (*ib\_insync.objects.AccountValue* attribute), 88  
`value` (*ib\_insync.objects.PnLSingle* attribute), 86  
`vega` (*ib\_insync.objects.OptionComputation* attribute), 91  
`verifyAndAuthMessage()` (*ib\_insync.client.Client* method), 33  
`verifyAndAuthRequest()` (*ib\_insync.client.Client* method), 33  
`verifyMessage()` (*ib\_insync.client.Client* method), 33  
`verifyRequest()` (*ib\_insync.client.Client* method), 33  
`volatility` (*ib\_insync.order.Order* attribute), 38  
`volatilityType` (*ib\_insync.order.Order* attribute), 38  
`volume` (*ib\_insync.objects.BarData* attribute), 80  
`volume` (*ib\_insync.objects.RealTimeBar* attribute), 80  
`volume` (*ib\_insync.order.VolumeCondition* attribute), 51  
`volume` (*ib\_insync.ticker.Bar* attribute), 74  
`volume` (*ib\_insync.ticker.Ticker* attribute), 70  
`VolumeBars` (class in *ib\_insync.ticker*), 74

`volumebars()` (*ib\_insync.ticker.Tickfilter* method), 73  
`VolumeCondition` (class in *ib\_insync.order*), 51  
`volumeRate` (*ib\_insync.ticker.Ticker* attribute), 71  
`vwap` (*ib\_insync.ticker.Ticker* attribute), 70

## W

`waitOnUpdate()` (*ib\_insync.ib.IB* method), 9  
`waitUntil()` (*ib\_insync.ib.IB* static method), 8  
`waitUntil()` (in module *ib\_insync.util*), 96  
`waitUntilAsync()` (in module *ib\_insync.util*), 97  
`wap` (*ib\_insync.objects.RealTimeBar* attribute), 80  
`warningText` (*ib\_insync.order.OrderState* attribute), 45  
`Warrant` (class in *ib\_insync.contract*), 62  
`Watchdog` (class in *ib\_insync.ibcontroller*), 101  
`whatIf` (*ib\_insync.order.Order* attribute), 39  
`whatIfOrder()` (*ib\_insync.ib.IB* method), 13  
`whatIfOrderAsync()` (*ib\_insync.ib.IB* method), 27  
`whatToShow` (*ib\_insync.objects.BarDataList* attribute), 93  
`whatToShow` (*ib\_insync.objects.RealTimeBarList* attribute), 94  
`whyHeld` (*ib\_insync.order.OrderStatus* attribute), 44  
`WshEventData` (class in *ib\_insync.objects*), 87

## Y

`yield_` (*ib\_insync.objects.CommissionReport* attribute), 78  
`yieldRedemptionDate` (*ib\_insync.objects.CommissionReport* attribute), 78