
ib_insync

Release 0.9.71

Ewald de Wit

Sep 30, 2022

CONTENTS

1	Introduction	3
1.1	Installation	3
1.2	Example	3
1.3	Documentation	4
1.4	Discussion	4
1.5	Consultancy & Development	4
1.6	Disclaimer	5
2	API docs	7
2.1	IB	7
2.2	Client	30
2.3	Order	35
2.4	Contract	53
2.5	Ticker	69
2.6	Objects	76
2.7	Utilities	95
2.8	FlexReport	98
2.9	IBC	98
2.10	IBController	101
2.11	Watchdog	102
3	Notebooks	105
4	Code recipes	107
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	Dividends	110
4.10	Fundamental ratios	110
4.11	Async streaming ticks	110
4.12	Integration with PyQt5 or PySide2	111
4.13	Integration with Tkinter	112
4.14	Integration with PyGame	112
5	Source code	113

6	Changelog	115
6.1	0.9	115
6.2	0.8	126
6.3	0.7	129
6.4	0.6	130
7	Links	131
8	Introduction	133
8.1	Installation	133
8.2	Example	133
8.3	Documentation	134
8.4	Discussion	134
8.5	Consultancy & Development	134
8.6	Disclaimer	135
	Python Module Index	137
	Index	139

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
```

For Python 3.6 install the `dataclasses` package as well (newer Python versions already have it):

```
pip install dataclasses
```

Requirements:

- Python 3.6 or higher;
- A running TWS or IB Gateway application (version 972 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)
```

(continues on next page)

(continued from previous page)

```
contract = Forex('EURUSD')
bars = ib.reqHistoricalData(
    contract, endDateTime='', durationStr='30 D',
    barSizeSetting='1 hour', whatToShow='MIDPOINT', useRTH=True)

# convert to pandas dataframe:
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 Consultancy & Development

IB-insync offers an easy entry into building automated trading systems for both individual traders and fintech companies. However, to get the most out of it is not a trivial matter and is beyond the reach of most developers.

If you need expert help, you can contact me. This can be for a small project, such as fixing something in your own code, or it can be creating an entire new trading infrastructure. Please provide enough details so that I can assess both the feasibility and the scope. Many folks worry about having to provide their ‘secret sauce’, but that is never necessary (although you’re perfectly welcome to send that as well!)

1.6 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>

Release 0.9.71.

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** (*pytz.timezone*) – 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.

- `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) 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', 'errorEvent',
'timeoutEvent')
```

RequestTimeout: float = 0

RaiseRequestErrors: bool = False

MaxSyncedSubAccounts: int = 50

TimezoneTWS = None

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

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.

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 `EventHandle`.

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()

List of portfolio items of the default account.

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

`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

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*)

Cancel the order and return the Trade it belongs to.

Parameters

order (*Order*) – The order to be canceled.

Return type*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/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[Order]`

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[Order]`

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

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

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

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

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

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; endDateTime 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*, *endDateTime=""*, *useRTH=True*)

Request historical schedule.

This method is blocking.

Parameters

- **contract** (*Contract*) – Contract of interest.
- **numDays** (*int*) – Number of days.
- **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.
- **useRTH** (*bool*) – If True then show schedule for Regular Trading Hours, if False then for extended hours.

Return type

HistoricalSchedule

reqHistoricalTicks(*contract*, *startDateTime*, *endDateTime*, *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

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.
- **endDateTime** (*Union[str, date]*) – One of `startDateTime` or `endDateTime` 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

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=None*)

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

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** (*Optional[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**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**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

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

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

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

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, optPrOptions=[]*)

Calculate the option price given the volatility.

This method is blocking.

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=None)

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** (*Optional[List[TagValue]]*) – Unknown.

Return type

NewsArticle

reqHistoricalNews(conId, providerCodes, startDateTime, endDateTime, totalResults, historicalNewsOptions=None)

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** (*Optional[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.

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

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[Order]]`

reqAllOpenOrdersAsync()

Return type
`Awaitable[List[Order]]`

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]`

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

Return type

Awaitable[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=None*)

Return type

Optional[HistoricalNews]

`async requestFAAsync(faDataType)`

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 = 167

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+.

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*)

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*)

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, opaqueIsvKey*)

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, conId*)

cancelWshEventData(*reqId*)

reqUserInfo(*reqId*)

2.3 Order

Order types used by Interactive Brokers.

```

class ib_insync.order.Order(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:
    ~typing.List[~ib_insync.contract.TagValue] = <factory>,
    smartComboRoutingParams: ~typing.List[~ib_insync.contract.TagValue] =
    <factory>, algoId: str = "", whatIf: bool = False, notHeld: bool = False,
    solicited: bool = False, modelCode: str = "", orderComboLegs:
    ~typing.List[~ib_insync.order.OrderComboLeg] = <factory>,
    orderMiscOptions: ~typing.List[~ib_insync.contract.TagValue] = <factory>,
    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:
    ~typing.List[~ib_insync.order.OrderCondition] = <factory>,
    conditionsCancelOrder: bool = False, conditionsIgnoreRth: bool = False,
    extOperator: str = "", softDollarTier: ~ib_insync.objects.SoftDollarTier =
    <factory>, cashQty: float = 1.7976931348623157e+308, hardDollarTier:
    str = "", mifid2DecisionAlgo: str = "", mifid2ExecutionTrader: str = "",
    mifid2ExecutionAlgo: str = "", dontUseAutoPriceForHedge: bool = False,
    isOmsContainer: bool = False, discretionaryUpToLimitPrice: bool = False,

```


Order for trading contracts.

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 = '0'
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
postToAts: int = 2147483647
advancedErrorOverride: 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.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***algoParams:** *List[TagValue]***smartComboRoutingParams:** *List[TagValue]***orderComboLegs:** *List[OrderComboLeg]***orderMiscOptions:** *List[TagValue]***conditions:** *List[OrderCondition]***softDollarTier:** *SoftDollarTier***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***algoParams:** *List[TagValue]***smartComboRoutingParams:** *List[TagValue]***orderComboLegs:** *List[OrderComboLeg]***orderMiscOptions:** *List[TagValue]***conditions:** *List[OrderCondition]*

softDollarTier: *SoftDollarTier*

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

algoParams: *List[TagValue]*

smartComboRoutingParams: *List[TagValue]*

orderComboLegs: *List[OrderComboLeg]*

orderMiscOptions: *List[TagValue]*

conditions: *List[OrderCondition]*

softDollarTier: *SoftDollarTier*

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

algoParams: List[TagValue]

smartComboRoutingParams: List[TagValue]

orderComboLegs: List[OrderComboLeg]

orderMiscOptions: List[TagValue]

conditions: List[OrderCondition]

softDollarTier: SoftDollarTier

```
class ib_insync.order.OrderStatus(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)
```

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[Set[str]] = {'ApiCancelled', 'Cancelled', 'Filled'}`

ActiveStates: `ClassVar[Set[str]] = {'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: 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 = "")
```

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: float = 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: ~ib_insync.contract.Contract = <factory>, order:
    ~ib_insync.order.Order = <factory>, orderStatus:
    ~ib_insync.order.OrderStatus = <factory>, fills:
    ~typing.List[~ib_insync.objects.Fill] = <factory>, log:
    ~typing.List[~ib_insync.objects.TradeLogEntry] = <factory>, advancedError:
    str = '')
```

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`)

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

```
contract: Contract
```

```
order: Order
```

```
orderStatus: OrderStatus
```

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

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

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

isActive()

True if eligible for execution, false otherwise.

isDone()

True if completely filled or cancelled, false otherwise.

filled()

Number of shares filled.

remaining()

Number of shares remaining to be filled.

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.BracketOrder`(*parent, takeProfit, stopLoss*)

Create new instance of `BracketOrder`(*parent, takeProfit, stopLoss*)

property `parent`

property `takeProfit`

property `stopLoss`

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: int = 1, conjunction: str = 'a', isMore: bool = True,
    price: float = 0.0, conId: int = 0, exch: str = "", triggerMethod: int = 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: int = 3, conjunction: str = 'a', isMore: bool = True, time:
    str = "")
```

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: int = 4, conjunction: str = 'a', isMore: bool = True,  
percent: int = 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: int = 5, conjunction: str = 'a', secType: str = "",
                                       exch: str = "", symbol: str = "")
```

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: int = 6, conjunction: str = 'a', isMore: bool = True,
                                       volume: int = 0, conId: int = 0, exch: str = "")
```

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: int = 7, conjunction: str = 'a', isMore: bool = True, changePercent: float = 0.0, conId: int = 0, exch: str = '')
```

```
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: 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 = "",
                                  comboLegsDescrip: str = "", comboLegs:
                                  ~typing.List[~ib_insync.contract.ComboLeg] = <factory>,
                                  deltaNeutralContract:
                                  ~typing.Optional[~ib_insync.contract.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
- 'COMDTY' = Commodity
- 'NEWS' = News
- 'FUND' = Mutual fund
- 'CRYPTO' = Crypto currency
- **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 = ''
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***comboLegs:** `List[ComboLeg]`**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

comboLegs: List[ComboLeg]

```
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`

comboLegs: `List[ComboLeg]`

```
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`

comboLegs: `List[ComboLeg]`

```
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

comboLegs: List[ComboLeg]

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`

comboLegs: `List[ComboLeg]`

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`

comboLegs: `List[ComboLeg]`

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

comboLegs: `List[ComboLeg]`

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

comboLegs: `List[ComboLeg]`

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

comboLegs: `List[ComboLeg]`

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`

comboLegs: `List[ComboLeg]`

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`

comboLegs: `List[ComboLeg]`

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`

comboLegs: `List[ComboLeg]`

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`

comboLegs: `List[ComboLeg]`

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

Create new instance of `TagValue`(tag, value)

property tag

property value

```
class ib_insync.contract.ComboLeg(conId: int = 0, ratio: int = 0, action: str = "", exchange: str = "",
                                  openClose: int = 0, shortSaleSlot: int = 0, designatedLocation: str = "",
                                  exemptCode: int = -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: int = 0, delta: float = 0.0, price: float = 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.ContractDetails(contract: Union[ib_insync.contract.Contract, NoneType] =
    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[ib_insync.contract.TagValue] = <factory>,
    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: int
    = 0, convertible: bool = False, maturity: str = "", issueDate: str
    = "", nextOptionDate: str = "", nextOptionType: str = "",
    nextOptionPartial: bool = False, notes: str = "")
```

```
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
puttable: bool = False
coupon: int = 0
convertible: bool = False
maturity: str = ''
issueDate: str = ''
nextOptionDate: str = ''
```

nextOptionType: `str` = ''

nextOptionPartial: `bool` = False

notes: `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.ContractDescription`(*contract: Union[ib_insync.contract.Contract, NoneType]*
= None, derivativeSecTypes: List[str] = <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: int, contractDetails: ib_insync.contract.ContractDetails, distance: str, benchmark: str, projection: str, legsStr: str*)

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: ~typing.Optional[~ib_insync.contract.Contract] = None, time:
    ~typing.Optional[~datetime.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:
    ~typing.Optional[~datetime.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:
    ~typing.Optional[~ib_insync.objects.Dividends] = None, fundamentalRatios:
    ~typing.Optional[~ib_insync.objects.FundamentalRatios] = None, ticks:
    ~typing.List[~ib_insync.objects.TickData] = <factory>, tickByTicks:
    ~typing.List[~typing.Union[~ib_insync.objects.TickByTickAllLast,
    ~ib_insync.objects.TickByTickBidAsk,
    ~ib_insync.objects.TickByTickMidPoint]] = <factory>, domBids:
    ~typing.List[~ib_insync.objects.DOMLevel] = <factory>, domAsks:
    ~typing.List[~ib_insync.objects.DOMLevel] = <factory>, domTicks:
    ~typing.List[~ib_insync.objects.MktDepthData] = <factory>, bidGreeks:
    ~typing.Optional[~ib_insync.objects.OptionComputation] = None,
    askGreeks: ~typing.Optional[~ib_insync.objects.OptionComputation] =
    None, lastGreeks: ~typing.Optional[~ib_insync.objects.OptionComputation]
    = None, modelGreeks:
    ~typing.Optional[~ib_insync.objects.OptionComputation] = None,
    auctionVolume: float = nan, auctionPrice: float = nan, auctionImbalance:
    float = nan, regulatoryImbalance: float = nan, bboExchange: str = "",
    snapshotPermissions: int = 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

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

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.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***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: Union[datetime.datetime, NoneType]*, *open: float = nan*, *high: float = nan*,
low: float = nan, *close: float = nan*, *volume: int = 0*, *count: int = 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.

2.6 Objects

Object hierarchy.

```
class ib_insync.objects.ScannerSubscription(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 = ")
```

```
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: str = "", val: str = "", displayName: str = "")`

`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: str = "", time: datetime.datetime = datetime.datetime(1970, 1, 1, 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)
```

execId: *str* = ''

time: *datetime* = datetime.datetime(1970, 1, 1, 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

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: str = "", commission: float = 0.0, currency: str = "",
                                         realizedPNL: float = 0.0, yield_: float = 0.0,
                                         yieldRedemptionDate: int = 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: int = 0, acctCode: str = "", time: str = "", symbol: str =
                                         "", secType: str = "", exchange: str = "", side: str = "")
```

```
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: Union[datetime.date, datetime.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)
```

```
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 = 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)
```

```
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: bool = False, pastLimit: bool = False, preOpen: bool = 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: bool = False, askPastHigh: bool = 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: bool = False, unreported: bool = 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: float = 0.0, count: int = 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: str = "", name: str = "")
```

```
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: str = "", secType: str = "", listingExch: str =  
                                                ", serviceDataType: str = "", aggGroup: int =  
                                                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: str = "", modelCode: str = "", dailyPnL: float = nan, unrealizedPnL: float = nan, realizedPnL: float = 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: datetime.datetime, status: str = "", message: str = "", errorCode: int = 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: str = "", modelCode: str = "", conId: int = 0, dailyPnL: float = nan, unrealizedPnL: float = nan, realizedPnL: float = nan, position: int = 0, value: float = 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: str = "", endDateTime: str = "", refDate: str =
    ")
```

```
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: str = "", endDateTime: str = "", timeZone: str = "", sessions: List[ib_insync.objects.HistoricalSession] = <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.AccountValue(account, tag, value, currency, modelCode)
```

Create new instance of AccountValue(account, tag, value, currency, modelCode)

```
property account
```

```
property tag
```

```
property value
```

```
property currency
```

```
property modelCode
```

```
class ib_insync.objects.TickData(time, tickType, price, size)
```

Create new instance of TickData(time, tickType, price, size)

```
property time
```

```
property tickType
```

```
property price
```

```
property size
```

```
class ib_insync.objects.HistoricalTick(time, price, size)
    Create new instance of HistoricalTick(time, price, size)
    property time
    property price
    property size

class ib_insync.objects.HistoricalTickBidAsk(time, tickAttribBidAsk, priceBid, priceAsk, sizeBid, sizeAsk)
    Create new instance of HistoricalTickBidAsk(time, tickAttribBidAsk, priceBid, priceAsk, sizeBid, sizeAsk)
    property time
    property tickAttribBidAsk
    property priceBid
    property priceAsk
    property sizeBid
    property sizeAsk

class ib_insync.objects.HistoricalTickLast(time, tickAttribLast, price, size, exchange, specialConditions)
    Create new instance of HistoricalTickLast(time, tickAttribLast, price, size, exchange, specialConditions)
    property time
    property tickAttribLast
    property price
    property size
    property exchange
    property specialConditions

class ib_insync.objects.TickByTickAllLast(tickType, time, price, size, tickAttribLast, exchange, specialConditions)
    Create new instance of TickByTickAllLast(tickType, time, price, size, tickAttribLast, exchange, specialConditions)
    property tickType
    property time
    property price
    property size
    property tickAttribLast
    property exchange
    property specialConditions
```

class `ib_insync.objects.TickByTickBidAsk`(*time, bidPrice, askPrice, bidSize, askSize, tickAttribBidAsk*)

Create new instance of `TickByTickBidAsk`(*time, bidPrice, askPrice, bidSize, askSize, tickAttribBidAsk*)

property `time`

property `bidPrice`

property `askPrice`

property `bidSize`

property `askSize`

property `tickAttribBidAsk`

class `ib_insync.objects.TickByTickMidPoint`(*time, midPoint*)

Create new instance of `TickByTickMidPoint`(*time, midPoint*)

property `time`

property `midPoint`

class `ib_insync.objects.MktDepthData`(*time, position, marketMaker, operation, side, price, size*)

Create new instance of `MktDepthData`(*time, position, marketMaker, operation, side, price, size*)

property `time`

property `position`

property `marketMaker`

property `operation`

property `side`

property `price`

property `size`

class `ib_insync.objects.DOMLevel`(*price, size, marketMaker*)

Create new instance of `DOMLevel`(*price, size, marketMaker*)

property `price`

property `size`

property `marketMaker`

class `ib_insync.objects.PriceIncrement`(*lowEdge, increment*)

Create new instance of `PriceIncrement`(*lowEdge, increment*)

property `lowEdge`

property `increment`

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*)

property contract

property position

property marketPrice

property marketValue

property averageCost

property unrealizedPNL

property realizedPNL

property account

class `ib_insync.objects.Position`(*account, contract, position, avgCost*)

Create new instance of Position(account, contract, position, avgCost)

property account

property contract

property position

property avgCost

class `ib_insync.objects.Fill`(*contract, execution, commissionReport, time*)

Create new instance of Fill(contract, execution, commissionReport, time)

property contract

property execution

property commissionReport

property time

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)

property tickAttrib

property impliedVol

property delta

property optPrice

property pvDividend

property gamma

property vega

property theta

property undPrice

```
class ib_insync.objects.OptionChain(exchange, underlyingConId, tradingClass, multiplier, expirations,  
strikes)  
    Create new instance of OptionChain(exchange, underlyingConId, tradingClass, multiplier, expirations, strikes)  
property exchange  
property underlyingConId  
property tradingClass  
property multiplier  
property expirations  
property strikes  
  
class ib_insync.objects.Dividends(past12Months, next12Months, nextDate, nextAmount)  
    Create new instance of Dividends(past12Months, next12Months, nextDate, nextAmount)  
property past12Months  
property next12Months  
property nextDate  
property nextAmount  
  
class ib_insync.objects.NewsArticle(articleType, articleText)  
    Create new instance of NewsArticle(articleType, articleText)  
property articleType  
property articleText  
  
class ib_insync.objects.HistoricalNews(time, providerCode, articleId, headline)  
    Create new instance of HistoricalNews(time, providerCode, articleId, headline)  
property time  
property providerCode  
property articleId  
property headline  
  
class ib_insync.objects.NewsTick(timeStamp, providerCode, articleId, headline, extraData)  
    Create new instance of NewsTick(timeStamp, providerCode, articleId, headline, extraData)  
property timeStamp  
property providerCode  
property articleId  
property headline  
property extraData  
  
class ib_insync.objects.NewsBulletin(msgId, msgType, message, origExchange)  
    Create new instance of NewsBulletin(msgId, msgType, message, origExchange)
```


property msgId

property msgType

property message

property origExchange

class `ib_insync.objects.FamilyCode`(*accountID, familyCodeStr*)

Create new instance of FamilyCode(accountID, familyCodeStr)

property accountID

property familyCodeStr

class `ib_insync.objects.SmartComponent`(*bitNumber, exchange, exchangeLetter*)

Create new instance of SmartComponent(bitNumber, exchange, exchangeLetter)

property bitNumber

property exchange

property exchangeLetter

class `ib_insync.objects.ConnectionStats`(*startTime, duration, numBytesRecv, numBytesSent, numMsgRecv, numMsgSent*)

Create new instance of ConnectionStats(startTime, duration, numBytesRecv, numBytesSent, numMsgRecv, numMsgSent)

property startTime

property duration

property numBytesRecv

property numBytesSent

property numMsgRecv

property numMsgSent

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: `Optional[Union[datetime, date, str]]`

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://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.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 `date-time.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(dt)`

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*)

Download and parse IB account statements via the Flex Web Service. https://guides.interactivebrokers.com/am/am/reports/flex_web_service_version_3.htm <https://guides.interactivebrokers.com/cp/cp.htm#am/reporting/flexqueries.htm>

Make sure to use a XML query (not CSV). A large query can take a few minutes. In the weekends the query servers can be down.

Download a report by giving a valid `token` and `queryId`, or load from file by giving a valid `path`.

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: int = 0, gateway: bool = False, tradingMode: str = "", twspath: str = "", twsSettingsPath: str = "", ibcPath: str = "", ibcIni: str = "", javaPath: str = "", userid: str = "", password: str = "", fixuserid: str = "", fixpassword: str = ""*)

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
- **twsettingsPath** (*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).

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 = ''`

`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 IBController

```
class ib_insync.ibcontroller.IBController(APP: str = 'TWS', TWS_MAJOR_VRSN: str = '969',
    TRADING_MODE: str = 'live', IBC_INI: str =
    '~/IBController/IBController.ini', IBC_PATH: str =
    '~/IBController', TWS_PATH: str = '~/Jts', LOG_PATH: str =
    '~/IBController/Logs', TWSUSERID: str = "",
    TWSPASSWORD: str = "", JAVA_PATH: str = "",
    TWS_CONFIG_PATH: str = "")
```

For new installations it is recommended to use IBC instead.

Programmatic control over starting and stopping TWS/Gateway using IBController (<https://github.com/ib-controller/ib-controller>).

On Windows the the proactor (or quamaash) event loop must have been set:

```
import asyncio
asyncio.set_event_loop(asyncio.ProactorEventLoop())
```

This is not intended to be run in a notebook.

```
APP: str = 'TWS'

TWS_MAJOR_VRSN: str = '969'

TRADING_MODE: str = 'live'

IBC_INI: str = '~/IBController/IBController.ini'

IBC_PATH: str = '~/IBController'

TWS_PATH: str = '~/Jts'

LOG_PATH: str = '~/IBController/Logs'

TWSUSERID: str = ''

TWSPASSWORD: str = ''

JAVA_PATH: str = ''

TWS_CONFIG_PATH: str = ''

start()
    Launch TWS/IBG.

stop()
    Cleanly shutdown TWS/IBG.

terminate()
    Terminate TWS/IBG.

async startAsync()

async stopAsync()

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

2.11 Watchdog

```
class ib_insync.ibcontroller.Watchdog(controller: Union[IBC, IBController], 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 = "",
    probeContract: Contract = Forex('EURUSD',
    exchange='IDEALPRO'), probeTimeout: float = 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** (*Union[IBC, IBController]*) – (required) IBC or IBController 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())

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: Union[IBC, IBController]
```

```
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 = ''`

`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 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)
```

4.10 Fundamental ratios

```
contract = Stock('IBM', 'SMART', 'USD')
ticker = ib.reqMktData(contract, '258')
ib.sleep(2)
print(ticker.fundamentalRatios)
```

4.11 Async streaming ticks

```
import asyncio

import ib_insync as ibi

class App:

    async def run(self):
        self.ib = ibi.IB()
        with await self.ib.connectAsync():
            contracts = [
                ibi.Stock(symbol, 'SMART', 'USD')
                for symbol in ['AAPL', 'TSLA', 'AMD', 'INTC']]
            for contract in contracts:
                self.ib.reqMktData(contract)

            async for tickers in self.ib.pendingTickersEvent:
                for ticker in tickers:
                    print(ticker)

    def stop(self):
        self.ib.disconnect()

app = App()
try:
    asyncio.run(app.run())
```

(continues on next page)

(continued from previous page)

```
except (KeyboardInterrupt, SystemExit):
    app.stop()
```

4.12 Integration with PyQt5 or PySide2

	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.

4.13 Integration with Tkinter

To integrate with the Tkinter event loop, take a look at [this example app](#).

4.14 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 0.9.71

- [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.

6.1.2 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.

6.1.3 Version 0.9.69

- Fixed: [issue 403](#): Change validity test for whatIfOrder response.

6.1.4 Version 0.9.68

- Fixed: [issue 402](#): Downloading historical ticks for crypto currencies.

6.1.5 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.

6.1.6 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.

6.1.7 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.

6.1.8 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](#)).

6.1.9 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.

6.1.10 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.

6.1.11 Version 0.9.61

- `Ticker.marketDataType` added to indicate the delayed/frozen status of the `reqMktData` ticks.

6.1.12 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!](#)

6.1.13 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`

6.1.14 Version 0.9.58

- PR #196 treats error 492 as a warning so that scanner results can still be used.

6.1.15 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.

6.1.16 Version 0.9.56

- Fix bug #178: `Order.totalQuantity` is now float.

6.1.17 Version 0.9.55

- Sphinx update for documentation.

6.1.18 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.

6.1.19 Version 0.9.53

- Fix bug #155 (IB.commissionReportEvent not firing).
- Help editors with the code completion for `Events`.

6.1.20 Version 0.9.52

- Fix `Client.exerciseOptions` (bug #152).

6.1.21 Version 0.9.51

- Fix `ib.placeOrder` for older TWS/gateway versions.
- Better handling of unclean disconnects.

6.1.22 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.

6.1.23 Version 0.9.49

- `ib.reqCompletedOrders()` request added (requires TWS/gateway ≥ 976). Completed orders are automatically synced on connect and are available from `ib.trades()`, complete with fills and commission info.
- Fixed bug #144.

6.1.24 Version 0.9.48

- `Ticker.halted` field added.
- `Client.reqFundamentalData` fixed.

6.1.25 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.

6.1.26 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.

6.1.27 Version 0.9.45

- `Event.merge()` added.
- `TagValue` serialization fixed.

6.1.28 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!

6.1.29 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.

6.1.30 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.

6.1.31 Version 0.9.41

- Fixed bug #117.
- Fixed order modifications with TWS/gateway 974.

6.1.32 Version 0.9.40

- `Ticker.fundamentalRatios` added (for use with generic tick 258).
- Fixed `reqHistoricalTicks` with MIDPOINT.

6.1.33 Version 0.9.39

- Handle partially filled dividend data.
- Use `secType='WAR'` for warrants.

6.1.34 Version 0.9.38

- `ibapi v97.4` is now required.
- fixed `tickByTick` wrappers.

6.1.35 Version 0.9.37

- Backward compatibility with older `ibapi` restored.

6.1.36 Version 0.9.36

- Compatibility with `ibapi v974`.
- `Client.setConnectOptions()` added (for PACEAPI).

6.1.37 Version 0.9.35

- `Ticker.hasBidAsk()` added.
- `IB.newsBulletinEvent` added.
- Various small fixes.

6.1.38 Version 0.9.34

- Old event system (`ib.setCallback`) removed.
- Compatibility fix with previous `ibapi` version.

6.1.39 Version 0.9.33

- Market scanner subscription improved.
- `IB.scannerDataEvent` now emits the full list of `ScanData`.
- `ScanDataList` added.

6.1.40 Version 0.9.32

- Autocompletion with Jedi plugin as used in Spyder and VS Code working again.

6.1.41 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.

6.1.42 Version 0.9.30

- `Watchdog.stop()` will not trigger restart now.
- Fixed bug #93.

6.1.43 Version 0.9.29

- `util.patchAsyncio()` updated for Python 3.7.

6.1.44 Version 0.9.28

- `IB.RequestTimeout` added.
- `util.schedule()` accepts tz-aware datetimes now.
- Let `client.disconnect()` complete when no event loop is running.

6.1.45 Version 0.9.27

- Fixed bug #77.

6.1.46 Version 0.9.26

- PR #74 merged (`ib.reqCurrentTime()` method added).
- Fixed bug with order error handling.

6.1.47 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()`.

6.1.48 Version 0.9.24

- `ticker.futuresOpenInterest` added.
- `execution.time` was string, is now parsed to UTC datetime.
- `ib.reqMarketRule()` request added.

6.1.49 Version 0.9.23

- Compatibility with Tornado 5 as used in new Jupyter notebook server.

6.1.50 Version 0.9.22

- updated `ib.reqNewsArticle` and `ib.reqHistoricalNews` to `ibapi v9.73.07`.

6.1.51 Version 0.9.21

- updated `ib.reqTickByTickData()` signature to `ibapi v9.73.07` while keeping backward compatibility.

6.1.52 Version 0.9.20

- Fixed watchdog bug.

6.1.53 Version 0.9.19

- Don't overwrite `exchange='SMART'` in `qualifyContracts`.

6.1.54 Version 0.9.18

- Merged PR #65 (Fix misnamed event).

6.1.55 Version 0.9.17

- New IB events `disconnectedEvent`, `newOrderEvent`, `orderModifyEvent` and `cancelOrderEvent`.
- Watchdog improvements.

6.1.56 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.

6.1.57 Version 0.9.15

- Fixed bug in default order conditions.
- Fixed regression from v0.9.13 in `placeOrder`.

6.1.58 Version 0.9.14

- Fixed `orderStatus` callback regression.

6.1.59 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.

6.1.60 Version 0.9.12

- Run `IBC` and `IBController` directly instead of via shell.

6.1.61 Version 0.9.11

- Fixed bug when collecting ticks using `ib.waitOnUpdate()`.
- Added `ContFuture` class (continuous futures).
- Added `Ticker.midpoint()`.

6.1.62 Version 0.9.10

- `ib.accountValues()` fixed for use with multiple accounts.

6.1.63 Version 0.9.9

- Fixed issue #57

6.1.64 Version 0.9.8

- Fix for `ib.reqPnLSingle()`.

6.1.65 Version 0.9.7

- Profit and Loss (PnL) functionality added.

6.1.66 Version 0.9.6

- IBC added.
- PR #53 (delayed greeks) merged.
- `Ticker.futuresOpenInterest` field removed.

6.1.67 Version 0.9.5

- Fixed canceling bar and tick subscriptions.

6.1.68 Version 0.9.4

- Fixed issue #49.

6.1.69 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.

6.1.70 Version 0.9.2

- Historical ticks and realtime bars now return time in UTC.

6.1.71 Version 0.9.1

- `IBController` added.
- `openOrder` callback added.
- default arguments for `ib.connect()` and `ib.reqMktData()`.

6.1.72 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
```

For Python 3.6 install the `dataclasses` package as well (newer Python versions already have it):

```
pip install dataclasses
```

Requirements:

- Python 3.6 or higher;
- A running TWS or IB Gateway application (version 972 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)
```

(continues on next page)

(continued from previous page)

```

contract = Forex('EURUSD')
bars = ib.reqHistoricalData(
    contract, endDateTime='', durationStr='30 D',
    barSizeSetting='1 hour', whatToShow='MIDPOINT', useRTH=True)

# convert to pandas dataframe:
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 Consultancy & Development

IB-insync offers an easy entry into building automated trading systems for both individual traders and fintech companies. However, to get the most out of it is not a trivial matter and is beyond the reach of most developers.

If you need expert help, you can contact me. This can be for a small project, such as fixing something in your own code, or it can be creating an entire new trading infrastructure. Please provide enough details so that I can assess both the feasibility and the scope. Many folks worry about having to provide their ‘secret sauce’, but that is never necessary (although you’re perfectly welcome to send that as well!)

8.6 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>

PYTHON MODULE INDEX

i

`ib_insync.client`, 30
`ib_insync.contract`, 53
`ib_insync.flexreport`, 98
`ib_insync.ib`, 7
`ib_insync.objects`, 76
`ib_insync.order`, 35
`ib_insync.ticker`, 69
`ib_insync.util`, 95

A

- abovePrice (*ib_insync.objects.ScannerSubscription attribute*), 76
- aboveVolume (*ib_insync.objects.ScannerSubscription attribute*), 76
- account (*ib_insync.ibcontroller.Watchdog attribute*), 104
- account (*ib_insync.objects.AccountValue property*), 88
- account (*ib_insync.objects.PnL attribute*), 85
- account (*ib_insync.objects.PnLSingle attribute*), 86
- account (*ib_insync.objects.PortfolioItem property*), 91
- account (*ib_insync.objects.Position property*), 91
- account (*ib_insync.order.Order attribute*), 39
- accountID (*ib_insync.objects.FamilyCode property*), 93
- accountSummary() (*ib_insync.ib.IB method*), 11
- accountSummaryAsync() (*ib_insync.ib.IB method*), 27
- AccountValue (*class in ib_insync.objects*), 88
- accountValues() (*ib_insync.ib.IB method*), 11
- acctCode (*ib_insync.objects.ExecutionFilter attribute*), 79
- acctNumber (*ib_insync.objects.Execution attribute*), 78
- action (*ib_insync.contract.ComboLeg attribute*), 65
- action (*ib_insync.order.Order attribute*), 37
- activeStartTime (*ib_insync.order.Order attribute*), 37
- ActiveStates (*ib_insync.order.OrderStatus attribute*), 45
- 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*), 67
- aggGroup (*ib_insync.objects.DepthMktDataDescription attribute*), 85
- algoId (*ib_insync.order.Order attribute*), 39
- algoParams (*ib_insync.order.LimitOrder attribute*), 42
- algoParams (*ib_insync.order.MarketOrder attribute*), 42
- algoParams (*ib_insync.order.Order attribute*), 39
- algoParams (*ib_insync.order.StopLimitOrder attribute*), 44
- algoParams (*ib_insync.order.StopOrder attribute*), 43
- 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
- APP (*ib_insync.ibcontroller.IBController attribute*), 101
- appStartupTime (*ib_insync.ibcontroller.Watchdog attribute*), 103
- appTimeout (*ib_insync.ibcontroller.Watchdog attribute*), 103
- articleId (*ib_insync.objects.HistoricalNews property*), 92
- articleId (*ib_insync.objects.NewsTick property*), 92
- articleText (*ib_insync.objects.NewsArticle property*), 92
- articleType (*ib_insync.objects.NewsArticle property*), 92
- ask (*ib_insync.ticker.Ticker attribute*), 71
- askExchange (*ib_insync.ticker.Ticker attribute*), 71
- askGreeks (*ib_insync.ticker.Ticker attribute*), 72
- askPastHigh (*ib_insync.objects.TickAttribBidAsk attribute*), 82
- askPrice (*ib_insync.objects.TickByTickBidAsk property*), 90
- asks() (*ib_insync.ticker.TickerUpdateEvent method*), 74
- askSize (*ib_insync.objects.TickByTickBidAsk property*), 90
- askSize (*ib_insync.ticker.Ticker attribute*), 71
- askYield (*ib_insync.ticker.Ticker attribute*), 71
- auctionImbalance (*ib_insync.ticker.Ticker attribute*), 73

auctionPrice (*ib_insync.ticker.Ticker* attribute), 73
auctionStrategy (*ib_insync.order.Order* attribute), 38
auctionVolume (*ib_insync.ticker.Ticker* attribute), 73
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* property), 91
averageOptionVolumeAbove (*ib_insync.objects.ScannerSubscription* attribute), 77
avgCost (*ib_insync.objects.Position* property), 91
avgFillPrice (*ib_insync.order.OrderStatus* attribute), 44
avgPrice (*ib_insync.objects.Execution* attribute), 78
avOptionVolume (*ib_insync.ticker.Ticker* attribute), 72
avVolume (*ib_insync.ticker.Ticker* attribute), 72

B

Bag (class in *ib_insync.contract*), 63
Bar (class in *ib_insync.ticker*), 75
barCount (*ib_insync.objects.BarData* attribute), 80
BarData (class in *ib_insync.objects*), 80
BarDataList (class in *ib_insync.objects*), 93
BarList (class in *ib_insync.ticker*), 75
barplot() (in module *ib_insync.util*), 95
bars (*ib_insync.ticker.TickBars* attribute), 75
bars (*ib_insync.ticker.TimeBars* 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), 73
belowPrice (*ib_insync.objects.ScannerSubscription* attribute), 76
benchmark (*ib_insync.contract.ScanData* attribute), 69
bid (*ib_insync.ticker.Ticker* attribute), 71
bidasks() (*ib_insync.ticker.TickerUpdateEvent* method), 74
bidExchange (*ib_insync.ticker.Ticker* attribute), 71
bidGreeks (*ib_insync.ticker.Ticker* attribute), 72
bidPastLow (*ib_insync.objects.TickAttribBidAsk* attribute), 82
bidPrice (*ib_insync.objects.TickByTickBidAsk* property), 90
bids() (*ib_insync.ticker.TickerUpdateEvent* method), 74
bidSize (*ib_insync.objects.TickByTickBidAsk* property), 90
bidSize (*ib_insync.ticker.Ticker* attribute), 71
bidYield (*ib_insync.ticker.Ticker* attribute), 71

bitNumber (*ib_insync.objects.SmartComponent* property), 93
blockOrder (*ib_insync.order.Order* attribute), 37
Bond (class in *ib_insync.contract*), 61
bondType (*ib_insync.contract.ContractDetails* attribute), 67
BracketOrder (class in *ib_insync.order*), 48
bracketOrder() (*ib_insync.ib.IB* method), 14

C

calculateImpliedVolatility() (*ib_insync.client.Client* method), 33
calculateImpliedVolatility() (*ib_insync.ib.IB* method), 24
calculateImpliedVolatilityAsync() (*ib_insync.ib.IB* method), 29
calculateOptionPrice() (*ib_insync.client.Client* method), 33
calculateOptionPrice() (*ib_insync.ib.IB* method), 25
calculateOptionPriceAsync() (*ib_insync.ib.IB* method), 29
callable (*ib_insync.contract.ContractDetails* attribute), 67
callOpenInterest (*ib_insync.ticker.Ticker* attribute), 72
callVolume (*ib_insync.ticker.Ticker* attribute), 72
canAutoExecute (*ib_insync.objects.TickAttrib* attribute), 82
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), 19
Cancelled (*ib_insync.order.OrderStatus* attribute), 44
cancelMktData() (*ib_insync.client.Client* method), 32
cancelMktData() (*ib_insync.ib.IB* method), 22
cancelMktDepth() (*ib_insync.client.Client* method), 32
cancelMktDepth() (*ib_insync.ib.IB* method), 23
cancelNewsBulletins() (*ib_insync.client.Client* method), 32

cancelNewsBulletins() (*ib_insync.ib*.IB method), 26
cancelOrder() (*ib_insync.client*.Client method), 32
cancelOrder() (*ib_insync.ib*.IB method), 15
cancelPnL() (*ib_insync.client*.Client method), 34
cancelPnL() (*ib_insync.ib*.IB method), 17
cancelPnLSingle() (*ib_insync.client*.Client method), 34
cancelPnLSingle() (*ib_insync.ib*.IB method), 17
cancelPositions() (*ib_insync.client*.Client method), 33
cancelPositionsMulti() (*ib_insync.client*.Client method), 34
cancelRealTimeBars() (*ib_insync.client*.Client method), 33
cancelRealTimeBars() (*ib_insync.ib*.IB method), 18
cancelScannerSubscription() (*ib_insync.client*.Client method), 33
cancelScannerSubscription() (*ib_insync.ib*.IB method), 24
cancelTickByTickData() (*ib_insync.client*.Client method), 34
cancelTickByTickData() (*ib_insync.ib*.IB method), 22
cancelWshEventData() (*ib_insync.client*.Client method), 35
cancelWshMetaData() (*ib_insync.client*.Client method), 34
cashQty (*ib_insync.order*.Order attribute), 40
category (*ib_insync.contract*.ContractDetails attribute), 66
CFD (class in *ib_insync.contract*), 60
changePercent (*ib_insync.order*.PercentChangeCondition attribute), 52
chartOptions (*ib_insync.objects*.BarDataList attribute), 94
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), 103
clientId (*ib_insync.objects*.Execution attribute), 78
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), 81
close (*ib_insync.ticker*.Bar attribute), 75
close (*ib_insync.ticker*.Ticker attribute), 71
code (*ib_insync.objects*.NewsProvider attribute), 84
ComboLeg (class in *ib_insync.contract*), 64
comboLegs (*ib_insync.contract*.Bag attribute), 64
comboLegs (*ib_insync.contract*.Bond attribute), 61
comboLegs (*ib_insync.contract*.CFD attribute), 60
comboLegs (*ib_insync.contract*.Commodity attribute), 61
comboLegs (*ib_insync.contract*.ContFuture attribute), 58
comboLegs (*ib_insync.contract*.Contract attribute), 55
comboLegs (*ib_insync.contract*.Crypto attribute), 64
comboLegs (*ib_insync.contract*.Forex attribute), 59
comboLegs (*ib_insync.contract*.Future attribute), 58
comboLegs (*ib_insync.contract*.FuturesOption attribute), 62
comboLegs (*ib_insync.contract*.Index attribute), 60
comboLegs (*ib_insync.contract*.MutualFund attribute), 63
comboLegs (*ib_insync.contract*.Option attribute), 57
comboLegs (*ib_insync.contract*.Stock attribute), 56
comboLegs (*ib_insync.contract*.Warrant attribute), 63
comboLegsDescrip (*ib_insync.contract*.Contract attribute), 55
commission (*ib_insync.objects*.CommissionReport attribute), 79
commission (*ib_insync.order*.OrderState attribute), 46
commissionCurrency (*ib_insync.order*.OrderState attribute), 46
CommissionReport (class in *ib_insync.objects*), 79
commissionReport (*ib_insync.objects*.Fill property), 91
Commodity (class in *ib_insync.contract*), 60
completedStatus (*ib_insync.order*.OrderState attribute), 46
completedTime (*ib_insync.order*.OrderState attribute), 46
conditions (*ib_insync.order*.LimitOrder attribute), 42
conditions (*ib_insync.order*.MarketOrder attribute), 42
conditions (*ib_insync.order*.Order attribute), 40
conditions (*ib_insync.order*.StopLimitOrder attribute), 44
conditions (*ib_insync.order*.StopOrder attribute), 43
conditionsCancelOrder (*ib_insync.order*.Order attribute), 40
conditionsIgnoreRth (*ib_insync.order*.Order attribute), 40
condType (*ib_insync.order*.ExecutionCondition attribute), 51
condType (*ib_insync.order*.MarginCondition attribute), 50
condType (*ib_insync.order*.PercentChangeCondition attribute), 52
condType (*ib_insync.order*.PriceCondition attribute), 49
condType (*ib_insync.order*.TimeCondition attribute), 49
condType (*ib_insync.order*.VolumeCondition attribute), 51
conId (*ib_insync.contract*.ComboLeg attribute), 65
conId (*ib_insync.contract*.Contract attribute), 54
conId (*ib_insync.contract*.DeltaNeutralContract attribute), 65
conId (*ib_insync.objects*.PnLSingle attribute), 86

- conId (*ib_insync.order.PercentChangeCondition* attribute), 52
 - conId (*ib_insync.order.PriceCondition* attribute), 49
 - conId (*ib_insync.order.VolumeCondition* attribute), 51
 - conjunction (*ib_insync.order.ExecutionCondition* attribute), 51
 - conjunction (*ib_insync.order.MarginCondition* attribute), 50
 - conjunction (*ib_insync.order.PercentChangeCondition* attribute), 52
 - conjunction (*ib_insync.order.PriceCondition* attribute), 49
 - 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), 9
 - 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), 103
 - ContFuture (*class in ib_insync.contract*), 58
 - continuousUpdate (*ib_insync.order.Order* attribute), 39
 - Contract (*class in ib_insync.contract*), 53
 - contract (*ib_insync.contract.ContractDescription* attribute), 68
 - contract (*ib_insync.contract.ContractDetails* attribute), 66
 - contract (*ib_insync.objects.BarDataList* attribute), 93
 - contract (*ib_insync.objects.Fill* property), 91
 - contract (*ib_insync.objects.PortfolioItem* property), 90
 - contract (*ib_insync.objects.Position* property), 91
 - contract (*ib_insync.objects.RealTimeBarList* attribute), 94
 - contract (*ib_insync.order.Trade* attribute), 47
 - contract (*ib_insync.ticker.Ticker* attribute), 70
 - ContractDescription (*class in ib_insync.contract*), 68
 - ContractDetails (*class in ib_insync.contract*), 66
 - contractDetails (*ib_insync.contract.ScanData* attribute), 69
 - contractMonth (*ib_insync.contract.ContractDetails* attribute), 66
 - controller (*ib_insync.ibcontroller.Watchdog* attribute), 103
 - convertible (*ib_insync.contract.ContractDetails* attribute), 67
 - count (*ib_insync.objects.HistogramData* attribute), 83
 - count (*ib_insync.objects.RealTimeBar* attribute), 81
 - count (*ib_insync.ticker.Bar* attribute), 75
 - coupon (*ib_insync.contract.ContractDetails* attribute), 67
 - couponRateAbove (*ib_insync.objects.ScannerSubscription* attribute), 76
 - couponRateBelow (*ib_insync.objects.ScannerSubscription* attribute), 76
 - couponType (*ib_insync.contract.ContractDetails* attribute), 67
 - create() (*ib_insync.contract.Contract* static method), 55
 - createClass() (*ib_insync.order.OrderCondition* static method), 48
 - Crypto (*class in ib_insync.contract*), 64
 - cumQty (*ib_insync.objects.Execution* attribute), 78
 - currency (*ib_insync.contract.Contract* attribute), 55
 - currency (*ib_insync.objects.AccountValue* property), 88
 - currency (*ib_insync.objects.CommissionReport* attribute), 79
 - cusip (*ib_insync.contract.ContractDetails* attribute), 67
- ## D
- dailyPnL (*ib_insync.objects.PnL* attribute), 85
 - dailyPnL (*ib_insync.objects.PnLSingle* attribute), 86
 - 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), 80
 - delta (*ib_insync.contract.DeltaNeutralContract* attribute), 65
 - delta (*ib_insync.objects.OptionComputation* property), 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*), 65
 - deltaNeutralContract (*ib_insync.contract.Contract* attribute), 55
 - 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*), 84
 derivativeSecTypes (*ib_insync.contract.ContractDescription* attribute), 68
 descAppend (*ib_insync.contract.ContractDetails* attribute), 67
 designatedLocation (*ib_insync.contract.ComboLeg* attribute), 65
 designatedLocation (*ib_insync.order.Order* attribute), 38
 df() (*ib_insync.flexreport.FlexReport* method), 98
 df() (in module *ib_insync.util*), 95
 dict() (*ib_insync.contract.Bag* method), 63
 dict() (*ib_insync.contract.Bond* method), 61
 dict() (*ib_insync.contract.CFD* method), 60
 dict() (*ib_insync.contract.ComboLeg* method), 65
 dict() (*ib_insync.contract.Commodity* method), 60
 dict() (*ib_insync.contract.ContFuture* method), 58
 dict() (*ib_insync.contract.Contract* method), 55
 dict() (*ib_insync.contract.ContractDescription* method), 68
 dict() (*ib_insync.contract.ContractDetails* method), 68
 dict() (*ib_insync.contract.Crypto* method), 64
 dict() (*ib_insync.contract.DeltaNeutralContract* method), 65
 dict() (*ib_insync.contract.Forex* method), 59
 dict() (*ib_insync.contract.Future* method), 57
 dict() (*ib_insync.contract.FuturesOption* method), 62
 dict() (*ib_insync.contract.Index* method), 59
 dict() (*ib_insync.contract.MutualFund* method), 62
 dict() (*ib_insync.contract.Option* method), 56
 dict() (*ib_insync.contract.ScanData* method), 69
 dict() (*ib_insync.contract.Stock* method), 56
 dict() (*ib_insync.contract.Warrant* method), 63
 dict() (*ib_insync.ibcontroller.IBC* method), 100
 dict() (*ib_insync.ibcontroller.IBController* method), 102
 dict() (*ib_insync.ibcontroller.Watchdog* method), 104
 dict() (*ib_insync.objects.BarData* method), 80
 dict() (*ib_insync.objects.CommissionReport* method), 79
 dict() (*ib_insync.objects.DepthMktDataDescription* method), 85
 dict() (*ib_insync.objects.Execution* method), 78
 dict() (*ib_insync.objects.ExecutionFilter* method), 80
 dict() (*ib_insync.objects.HistogramData* method), 83
 dict() (*ib_insync.objects.HistoricalSchedule* method), 88
 dict() (*ib_insync.objects.HistoricalSession* method), 87
 dict() (*ib_insync.objects.NewsProvider* method), 84
 dict() (*ib_insync.objects.PnL* method), 85
 dict() (*ib_insync.objects.PnLSingle* method), 87
 dict() (*ib_insync.objects.RealTimeBar* method), 81
 dict() (*ib_insync.objects.ScannerSubscription* method), 77
 dict() (*ib_insync.objects.SoftDollarTier* method), 77
 dict() (*ib_insync.objects.TickAttrib* method), 82
 dict() (*ib_insync.objects.TickAttribBidAsk* method), 82
 dict() (*ib_insync.objects.TickAttribLast* method), 83
 dict() (*ib_insync.objects.TradeLogEntry* method), 86
 dict() (*ib_insync.order.ExecutionCondition* method), 51
 dict() (*ib_insync.order.LimitOrder* method), 41
 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), 46
 dict() (*ib_insync.order.OrderStatus* method), 45
 dict() (*ib_insync.order.PercentChangeCondition* method), 52
 dict() (*ib_insync.order.PriceCondition* method), 49
 dict() (*ib_insync.order.StopLimitOrder* method), 43
 dict() (*ib_insync.order.StopOrder* method), 43
 dict() (*ib_insync.order.TimeCondition* method), 50
 dict() (*ib_insync.order.Trade* method), 48
 dict() (*ib_insync.order.VolumeCondition* method), 51
 dict() (*ib_insync.ticker.Ticker* method), 73
 disconnect() (*ib_insync.client.Client* method), 32
 disconnect() (*ib_insync.ib.IB* method), 9
 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), 77
 displaySize (*ib_insync.order.Order* attribute), 37
 distance (*ib_insync.contract.ScanData* attribute), 69
 Dividends (class in *ib_insync.objects*), 92
 dividends (*ib_insync.ticker.Ticker* attribute), 72
 domAsks (*ib_insync.ticker.Ticker* attribute), 72
 domBids (*ib_insync.ticker.Ticker* attribute), 72
 DOMLevel (class in *ib_insync.objects*), 90
 domTicks (*ib_insync.ticker.Ticker* attribute), 72
 DoneStates (*ib_insync.order.OrderStatus* attribute), 45
 dontUseAutoPriceForHedge (*ib_insync.order.Order* attribute), 40

- download() (*ib_insync.flexreport.FlexReport* method), 98
- duration (*ib_insync.objects.ConnectionStats* property), 93
- duration (*ib_insync.order.Order* attribute), 41
- durationStr (*ib_insync.objects.BarDataList* attribute), 93
- DynamicObject (class in *ib_insync.objects*), 94
- ## E
- endDateTime (*ib_insync.objects.BarDataList* attribute), 93
- endDateTime (*ib_insync.objects.HistoricalSchedule* attribute), 88
- endDateTime (*ib_insync.objects.HistoricalSession* attribute), 87
- endTime (*ib_insync.objects.RealTimeBar* attribute), 81
- equityWithLoanAfter (*ib_insync.order.OrderState* attribute), 46
- equityWithLoanBefore (*ib_insync.order.OrderState* attribute), 45
- equityWithLoanChange (*ib_insync.order.OrderState* attribute), 45
- errorCode (*ib_insync.objects.TradeLogEntry* attribute), 86
- eTradeOnly (*ib_insync.order.Order* attribute), 38
- events (*ib_insync.client.Client* attribute), 30
- events (*ib_insync.ib.IB* attribute), 9
- events (*ib_insync.ibcontroller.Watchdog* attribute), 103
- events (*ib_insync.order.Trade* attribute), 47
- events (*ib_insync.ticker.Ticker* attribute), 70
- evMultiplier (*ib_insync.contract.ContractDetails* attribute), 67
- evMultiplier (*ib_insync.objects.Execution* attribute), 78
- evRule (*ib_insync.contract.ContractDetails* attribute), 67
- evRule (*ib_insync.objects.Execution* attribute), 78
- exch (*ib_insync.order.ExecutionCondition* attribute), 51
- exch (*ib_insync.order.PercentChangeCondition* attribute), 52
- exch (*ib_insync.order.PriceCondition* attribute), 49
- exch (*ib_insync.order.VolumeCondition* attribute), 51
- exchange (*ib_insync.contract.ComboLeg* attribute), 65
- exchange (*ib_insync.contract.Contract* attribute), 55
- exchange (*ib_insync.objects.DepthMktDataDescription* attribute), 84
- exchange (*ib_insync.objects.Execution* attribute), 78
- exchange (*ib_insync.objects.ExecutionFilter* attribute), 80
- exchange (*ib_insync.objects.HistoricalTickLast* property), 89
- exchange (*ib_insync.objects.OptionChain* property), 92
- exchange (*ib_insync.objects.SmartComponent* property), 93
- exchange (*ib_insync.objects.TickByTickAllLast* property), 89
- exchangeLetter (*ib_insync.objects.SmartComponent* property), 93
- excludeConvertible (*ib_insync.objects.ScannerSubscription* attribute), 77
- execId (*ib_insync.objects.CommissionReport* attribute), 79
- execId (*ib_insync.objects.Execution* attribute), 78
- Execution (class in *ib_insync.objects*), 78
- execution (*ib_insync.objects.Fill* property), 91
- ExecutionCondition (class in *ib_insync.order*), 51
- ExecutionFilter (class in *ib_insync.objects*), 79
- executions() (*ib_insync.ib.IB* method), 13
- exemptCode (*ib_insync.contract.ComboLeg* attribute), 65
- exemptCode (*ib_insync.order.Order* attribute), 38
- exerciseOptions() (*ib_insync.client.Client* method), 33
- exerciseOptions() (*ib_insync.ib.IB* method), 25
- expirations (*ib_insync.objects.OptionChain* property), 92
- extOperator (*ib_insync.order.Order* attribute), 40
- extract() (*ib_insync.flexreport.FlexReport* method), 98
- extraData (*ib_insync.objects.NewsTick* property), 92
- ## F
- faGroup (*ib_insync.order.Order* attribute), 38
- faMethod (*ib_insync.order.Order* attribute), 38
- FamilyCode (class in *ib_insync.objects*), 93
- familyCodeStr (*ib_insync.objects.FamilyCode* property), 93
- faPercentage (*ib_insync.order.Order* attribute), 38
- faProfile (*ib_insync.order.Order* attribute), 38
- Fill (class in *ib_insync.objects*), 91
- Filled (*ib_insync.order.OrderStatus* attribute), 44
- filled (*ib_insync.order.OrderStatus* attribute), 44
- filled() (*ib_insync.order.Trade* method), 47
- filledQuantity (*ib_insync.order.Order* attribute), 41
- fills (*ib_insync.order.Trade* attribute), 47
- fills() (*ib_insync.ib.IB* method), 13
- 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), 72
- Future (*class in ib_insync.contract*), 57
- futuresOpenInterest (*ib_insync.ticker.Ticker* attribute), 72
- FuturesOption (*class in ib_insync.contract*), 61
- ## G
- gamma (*ib_insync.objects.OptionComputation* property), 91
- gateway (*ib_insync.ibcontroller.IBC* attribute), 99
- getAccounts() (*ib_insync.client.Client* method), 31
- getLoop() (*in module ib_insync.util*), 97
- getReqId() (*ib_insync.client.Client* method), 31
- goodAfterTime (*ib_insync.order.Order* attribute), 37
- goodTillDate (*ib_insync.order.Order* attribute), 37
- ## H
- halted (*ib_insync.ticker.Ticker* attribute), 72
- hasBidAsk() (*ib_insync.ticker.Ticker* method), 73
- headline (*ib_insync.objects.HistoricalNews* property), 92
- headline (*ib_insync.objects.NewsTick* property), 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), 80
- high (*ib_insync.objects.RealTimeBar* attribute), 81
- high (*ib_insync.ticker.Bar* attribute), 75
- high (*ib_insync.ticker.Ticker* attribute), 71
- high13week (*ib_insync.ticker.Ticker* attribute), 71
- high26week (*ib_insync.ticker.Ticker* attribute), 71
- high52week (*ib_insync.ticker.Ticker* attribute), 71
- HistogramData (*class in ib_insync.objects*), 83
- HistoricalNews (*class in ib_insync.objects*), 92
- HistoricalSchedule (*class in ib_insync.objects*), 87
- HistoricalSession (*class in ib_insync.objects*), 87
- HistoricalTick (*class in ib_insync.objects*), 88
- HistoricalTickBidAsk (*class in ib_insync.objects*), 89
- HistoricalTickLast (*class in ib_insync.objects*), 89
- histVolatility (*ib_insync.ticker.Ticker* attribute), 72
- host (*ib_insync.ibcontroller.Watchdog* attribute), 103
- ## I
- IB (*class in ib_insync.ib*), 7
- ib (*ib_insync.ibcontroller.Watchdog* attribute), 103
- ib_insync.client
module, 30
- ib_insync.contract
module, 53
- ib_insync.flexreport
module, 98
- ib_insync.ib
module, 7
- ib_insync.objects
module, 76
- ib_insync.order
module, 35
- ib_insync.ticker
module, 69
- ib_insync.util
module, 95
- IBC (*class in ib_insync.ibcontroller*), 98
- IBC_INI (*ib_insync.ibcontroller.IBCController* attribute), 101
- IBC_PATH (*ib_insync.ibcontroller.IBCController* attribute), 101
- ibcIni (*ib_insync.ibcontroller.IBC* attribute), 100
- IbcLogLevel (*ib_insync.ibcontroller.IBC* attribute), 99
- IBController (*class in ib_insync.ibcontroller*), 101
- ibcPath (*ib_insync.ibcontroller.IBC* attribute), 100
- imbalanceOnly (*ib_insync.order.Order* attribute), 41
- impliedVol (*ib_insync.objects.OptionComputation* property), 91
- impliedVolatility (*ib_insync.ticker.Ticker* attribute), 72
- Inactive (*ib_insync.order.OrderStatus* attribute), 45
- includeExpired (*ib_insync.contract.Contract* attribute), 55
- increment (*ib_insync.objects.PriceIncrement* property), 90
- Index (*class in ib_insync.contract*), 59
- indexFuturePremium (*ib_insync.ticker.Ticker* attribute), 72
- industry (*ib_insync.contract.ContractDetails* attribute), 66
- 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), 76
- isActive() (*ib_insync.order.Trade* method), 47
- isConnected() (*ib_insync.client.Client* method), 31
- isConnected() (*ib_insync.ib.IB* method), 9
- isDone() (*ib_insync.order.Trade* method), 47
- isHashable() (*ib_insync.contract.Contract* method), 55
- isMore (*ib_insync.order.MarginCondition* attribute), 50
- isMore (*ib_insync.order.PercentChangeCondition* attribute), 52
- isMore (*ib_insync.order.PriceCondition* attribute), 49
- 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), 67

J

JAVA_PATH (*ib_insync.ibcontroller.IBController* attribute), 101
 javaPath (*ib_insync.ibcontroller.IBC* attribute), 100

K

keepUpToDate (*ib_insync.objects.BarDataList* attribute), 94

L

last (*ib_insync.ticker.Ticker* attribute), 71
 lastExchange (*ib_insync.ticker.Ticker* attribute), 71
 lastFillPrice (*ib_insync.order.OrderStatus* attribute), 44
 lastGreeks (*ib_insync.ticker.Ticker* attribute), 72
 lastLiquidity (*ib_insync.objects.Execution* attribute), 78
 lastSize (*ib_insync.ticker.Ticker* attribute), 71
 lastTradeDateOrContractMonth
 (*ib_insync.contract.Contract* attribute), 54
 lastTradeTime (*ib_insync.contract.ContractDetails* attribute), 67
 lastYield (*ib_insync.ticker.Ticker* attribute), 72
 legsStr (*ib_insync.contract.ScanData* attribute), 69
 LimitOrder (class in *ib_insync.order*), 41
 liquidation (*ib_insync.objects.Execution* attribute), 78
 liquidHours (*ib_insync.contract.ContractDetails* attribute), 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), 55
 locationCode (*ib_insync.objects.ScannerSubscription* attribute), 76
 log (*ib_insync.order.Trade* attribute), 47
 LOG_PATH (*ib_insync.ibcontroller.IBController* attribute), 101
 logToConsole() (in module *ib_insync.util*), 95
 logToFile() (in module *ib_insync.util*), 95
 longName (*ib_insync.contract.ContractDetails* attribute), 66
 loopUntil() (*ib_insync.ib.IB* method), 11
 low (*ib_insync.objects.BarData* attribute), 80
 low (*ib_insync.objects.RealTimeBar* attribute), 81

low (*ib_insync.ticker.Bar* attribute), 75
 low (*ib_insync.ticker.Ticker* attribute), 71
 low13week (*ib_insync.ticker.Ticker* attribute), 71
 low26week (*ib_insync.ticker.Ticker* attribute), 71
 low52week (*ib_insync.ticker.Ticker* attribute), 71
 lowEdge (*ib_insync.objects.PriceIncrement* property), 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), 11
 MarginCondition (class in *ib_insync.order*), 50
 marketCapAbove (*ib_insync.objects.ScannerSubscription* attribute), 76
 marketCapBelow (*ib_insync.objects.ScannerSubscription* attribute), 76
 marketDataType (*ib_insync.ticker.Ticker* attribute), 71
 marketMaker (*ib_insync.objects.DOMLevel* property), 90
 marketMaker (*ib_insync.objects.MktDepthData* property), 90
 marketName (*ib_insync.contract.ContractDetails* attribute), 66
 MarketOrder (class in *ib_insync.order*), 42
 marketPrice (*ib_insync.objects.PortfolioItem* property), 91
 marketPrice() (*ib_insync.ticker.Ticker* method), 73
 marketRuleIds (*ib_insync.contract.ContractDetails* attribute), 67
 marketValue (*ib_insync.objects.PortfolioItem* property), 91
 markPrice (*ib_insync.ticker.Ticker* attribute), 72
 maturity (*ib_insync.contract.ContractDetails* attribute), 67
 maturityDateAbove (*ib_insync.objects.ScannerSubscription* attribute), 76
 maturityDateBelow (*ib_insync.objects.ScannerSubscription* attribute), 76
 MaxClientVersion (*ib_insync.client.Client* attribute), 31
 maxCommission (*ib_insync.order.OrderState* attribute), 46
 MaxRequests (*ib_insync.client.Client* attribute), 30
 MaxSyncedSubAccounts (*ib_insync.ib.IB* attribute), 9
 mdSizeMultiplier (*ib_insync.contract.ContractDetails* attribute), 67
 message (*ib_insync.objects.NewsBulletin* property), 93
 message (*ib_insync.objects.TradeLogEntry* attribute), 86
 midPoint (*ib_insync.objects.TickByTickMidPoint* property), 90

- midpoint() (*ib_insync.ticker.Ticker* method), 73
- Midpoints (class in *ib_insync.ticker*), 75
- midpoints() (*ib_insync.ticker.TickerUpdateEvent* method), 74
- 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), 31
- minCommission (*ib_insync.order.OrderState* attribute), 46
- minQty (*ib_insync.order.Order* attribute), 37
- minSize (*ib_insync.contract.ContractDetails* attribute), 67
- minTick (*ib_insync.contract.ContractDetails* attribute), 66
- minTick (*ib_insync.ticker.Ticker* attribute), 71
- mktCapPrice (*ib_insync.order.OrderStatus* attribute), 44
- MktDepthData (class in *ib_insync.objects*), 90
- modelCode (*ib_insync.objects.AccountValue* property), 88
- modelCode (*ib_insync.objects.Execution* attribute), 78
- modelCode (*ib_insync.objects.PnL* attribute), 85
- modelCode (*ib_insync.objects.PnLSingle* attribute), 86
- modelCode (*ib_insync.order.Order* attribute), 40
- modelGreeks (*ib_insync.ticker.Ticker* attribute), 73
- module
- ib_insync.client*, 30
 - ib_insync.contract*, 53
 - ib_insync.flexreport*, 98
 - ib_insync.ib*, 7
 - ib_insync.objects*, 76
 - ib_insync.order*, 35
 - ib_insync.ticker*, 69
 - ib_insync.util*, 95
- monitorAsync() (*ib_insync.ibcontroller.IBC* method), 100
- monitorAsync() (*ib_insync.ibcontroller.IBCController* method), 101
- moodyRatingAbove (*ib_insync.objects.ScannerSubscription* attribute), 76
- moodyRatingBelow (*ib_insync.objects.ScannerSubscription* attribute), 76
- msgId (*ib_insync.objects.NewsBulletin* property), 92
- msgType (*ib_insync.objects.NewsBulletin* property), 93
- multiplier (*ib_insync.contract.Contract* attribute), 54
- multiplier (*ib_insync.objects.OptionChain* property), 92
- MutualFund (class in *ib_insync.contract*), 62
- N**
- name (*ib_insync.objects.NewsProvider* attribute), 84
- name (*ib_insync.objects.SoftDollarTier* attribute), 77
- 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), 13
- NewsProvider (class in *ib_insync.objects*), 84
- NewsTick (class in *ib_insync.objects*), 92
- newsTicks() (*ib_insync.ib.IB* method), 13
- next12Months (*ib_insync.objects.Dividends* property), 92
- nextAmount (*ib_insync.objects.Dividends* property), 92
- nextDate (*ib_insync.objects.Dividends* property), 92
- nextOptionDate (*ib_insync.contract.ContractDetails* attribute), 67
- nextOptionPartial (*ib_insync.contract.ContractDetails* attribute), 68
- nextOptionType (*ib_insync.contract.ContractDetails* attribute), 67
- nonDefaults() (*ib_insync.contract.Bag* method), 63
- nonDefaults() (*ib_insync.contract.Bond* method), 61
- nonDefaults() (*ib_insync.contract.CFD* method), 60
- nonDefaults() (*ib_insync.contract.ComboLeg* method), 65
- nonDefaults() (*ib_insync.contract.Commodity* method), 61
- nonDefaults() (*ib_insync.contract.ContFuture* method), 58
- nonDefaults() (*ib_insync.contract.Contract* method), 55
- nonDefaults() (*ib_insync.contract.ContractDescription* method), 68
- nonDefaults() (*ib_insync.contract.ContractDetails* method), 68
- nonDefaults() (*ib_insync.contract.Crypto* method), 64
- nonDefaults() (*ib_insync.contract.DeltaNeutralContract* method), 65
- nonDefaults() (*ib_insync.contract.Forex* method), 59
- nonDefaults() (*ib_insync.contract.Future* method), 57
- nonDefaults() (*ib_insync.contract.FuturesOption* method), 62
- nonDefaults() (*ib_insync.contract.Index* method), 59
- nonDefaults() (*ib_insync.contract.MutualFund* method), 62
- nonDefaults() (*ib_insync.contract.Option* method), 57
- nonDefaults() (*ib_insync.contract.ScanData* method), 69
- nonDefaults() (*ib_insync.contract.Stock* method), 56
- nonDefaults() (*ib_insync.contract.Warrant* method), 63
- nonDefaults() (*ib_insync.ibcontroller.IBC* method), 100

- nonDefaults() (*ib_insync.ibcontroller.IBController* method), 102
 - nonDefaults() (*ib_insync.ibcontroller.Watchdog* method), 104
 - nonDefaults() (*ib_insync.objects.BarData* method), 81
 - nonDefaults() (*ib_insync.objects.CommissionReport* method), 79
 - nonDefaults() (*ib_insync.objects.DepthMktDataDescription* method), 85
 - nonDefaults() (*ib_insync.objects.Execution* method), 78
 - nonDefaults() (*ib_insync.objects.ExecutionFilter* method), 80
 - nonDefaults() (*ib_insync.objects.HistogramData* method), 83
 - nonDefaults() (*ib_insync.objects.HistoricalSchedule* method), 88
 - nonDefaults() (*ib_insync.objects.HistoricalSession* method), 87
 - nonDefaults() (*ib_insync.objects.NewsProvider* method), 84
 - nonDefaults() (*ib_insync.objects.PnL* method), 85
 - nonDefaults() (*ib_insync.objects.PnLSingle* method), 87
 - nonDefaults() (*ib_insync.objects.RealTimeBar* method), 81
 - nonDefaults() (*ib_insync.objects.ScannerSubscription* method), 77
 - nonDefaults() (*ib_insync.objects.SoftDollarTier* method), 77
 - nonDefaults() (*ib_insync.objects.TickAttrib* method), 82
 - nonDefaults() (*ib_insync.objects.TickAttribBidAsk* method), 82
 - nonDefaults() (*ib_insync.objects.TickAttribLast* method), 83
 - nonDefaults() (*ib_insync.objects.TradeLogEntry* method), 86
 - nonDefaults() (*ib_insync.order.ExecutionCondition* method), 51
 - nonDefaults() (*ib_insync.order.LimitOrder* method), 41
 - 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), 46
 - nonDefaults() (*ib_insync.order.OrderStatus* method), 45
 - nonDefaults() (*ib_insync.order.PercentChangeCondition* method), 52
 - nonDefaults() (*ib_insync.order.PriceCondition* method), 49
 - nonDefaults() (*ib_insync.order.StopLimitOrder* method), 43
 - nonDefaults() (*ib_insync.order.StopOrder* method), 43
 - nonDefaults() (*ib_insync.order.TimeCondition* method), 50
 - nonDefaults() (*ib_insync.order.Trade* method), 48
 - nonDefaults() (*ib_insync.order.VolumeCondition* method), 52
 - nonDefaults() (*ib_insync.ticker.Ticker* method), 73
 - notes (*ib_insync.contract.ContractDetails* attribute), 68
 - notHeld (*ib_insync.order.Order* attribute), 40
 - numberOfRows (*ib_insync.objects.ScannerSubscription* attribute), 76
 - numBytesRecv (*ib_insync.objects.ConnectionStats* property), 93
 - numBytesSent (*ib_insync.objects.ConnectionStats* property), 93
 - numMsgRecv (*ib_insync.objects.ConnectionStats* property), 93
 - numMsgSent (*ib_insync.objects.ConnectionStats* property), 93
- O**
- ocaGroup (*ib_insync.order.Order* attribute), 37
 - ocaType (*ib_insync.order.Order* attribute), 37
 - on_source() (*ib_insync.ticker.Midpoints* method), 75
 - on_source() (*ib_insync.ticker.TickBars* method), 75
 - on_source() (*ib_insync.ticker.Tickfilter* method), 74
 - on_source() (*ib_insync.ticker.TimeBars* method), 75
 - oneCancelsAll() (*ib_insync.ib.IB* static method), 14
 - open (*ib_insync.objects.BarData* attribute), 80
 - open (*ib_insync.ticker.Bar* attribute), 75
 - open (*ib_insync.ticker.Ticker* attribute), 71
 - open_ (*ib_insync.objects.RealTimeBar* attribute), 81
 - openClose (*ib_insync.contract.ComboLeg* attribute), 65
 - openClose (*ib_insync.order.Order* attribute), 38
 - openOrders() (*ib_insync.ib.IB* method), 12
 - openTrades() (*ib_insync.ib.IB* method), 12
 - operation (*ib_insync.objects.MktDepthData* property), 90
 - Option (class in *ib_insync.contract*), 56
 - 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* property), 91
 - Or() (*ib_insync.order.OrderCondition* method), 48
 - Order (class in *ib_insync.order*), 35

- order (*ib_insync.order.Trade* attribute), 47
 OrderComboLeg (*class in ib_insync.order*), 46
 orderComboLegs (*ib_insync.order.LimitOrder* attribute), 42
 orderComboLegs (*ib_insync.order.MarketOrder* attribute), 42
 orderComboLegs (*ib_insync.order.Order* attribute), 40
 orderComboLegs (*ib_insync.order.StopLimitOrder* attribute), 44
 orderComboLegs (*ib_insync.order.StopOrder* attribute), 43
 OrderCondition (*class in ib_insync.order*), 48
 orderId (*ib_insync.objects.Execution* attribute), 78
 orderId (*ib_insync.order.Order* attribute), 37
 orderId (*ib_insync.order.OrderStatus* attribute), 44
 orderMiscOptions (*ib_insync.order.LimitOrder* attribute), 42
 orderMiscOptions (*ib_insync.order.MarketOrder* attribute), 42
 orderMiscOptions (*ib_insync.order.Order* attribute), 40
 orderMiscOptions (*ib_insync.order.StopLimitOrder* attribute), 44
 orderMiscOptions (*ib_insync.order.StopOrder* attribute), 43
 orderRef (*ib_insync.objects.Execution* attribute), 78
 orderRef (*ib_insync.order.Order* attribute), 37
 orders() (*ib_insync.ib.IB* method), 12
 OrderState (*class in ib_insync.order*), 45
 OrderStatus (*class in ib_insync.order*), 44
 orderStatus (*ib_insync.order.Trade* attribute), 47
 orderType (*ib_insync.order.Order* attribute), 37
 orderTypes (*ib_insync.contract.ContractDetails* attribute), 66
 origExchange (*ib_insync.objects.NewsBulletin* property), 93
 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), 59
 parent (*ib_insync.order.BracketOrder* property), 48
 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* property), 92
 pastLimit (*ib_insync.objects.TickAttrib* attribute), 82
 pastLimit (*ib_insync.objects.TickAttribLast* attribute), 83
 patchAsyncio() (*in module ib_insync.util*), 97
 peggedChangeAmount (*ib_insync.order.Order* attribute), 40
 PendingCancel (*ib_insync.order.OrderStatus* attribute), 44
 PendingSubmit (*ib_insync.order.OrderStatus* attribute), 44
 pendingTickers() (*ib_insync.ib.IB* method), 13
 percent (*ib_insync.order.MarginCondition* attribute), 50
 PercentChangeCondition (*class in ib_insync.order*), 52
 percentOffset (*ib_insync.order.Order* attribute), 37
 permId (*ib_insync.objects.Execution* attribute), 78
 permId (*ib_insync.order.Order* attribute), 37
 permId (*ib_insync.order.OrderStatus* attribute), 44
 placeOrder() (*ib_insync.client.Client* method), 32
 placeOrder() (*ib_insync.ib.IB* method), 15
 PnL (*class in ib_insync.objects*), 85
 pnl() (*ib_insync.ib.IB* method), 12
 PnLSingle (*class in ib_insync.objects*), 86
 pnlSingle() (*ib_insync.ib.IB* method), 12
 port (*ib_insync.ibcontroller.Watchdog* attribute), 103
 portfolio() (*ib_insync.ib.IB* method), 11
 PortfolioItem (*class in ib_insync.objects*), 90
 Position (*class in ib_insync.objects*), 91
 position (*ib_insync.objects.MktDepthData* property), 90
 position (*ib_insync.objects.PnLSingle* attribute), 86
 position (*ib_insync.objects.PortfolioItem* property), 91
 position (*ib_insync.objects.Position* property), 91
 positions() (*ib_insync.ib.IB* method), 12
 postToAts (*ib_insync.order.Order* attribute), 41
 preOpen (*ib_insync.objects.TickAttrib* attribute), 82
 PreSubmitted (*ib_insync.order.OrderStatus* attribute), 44
 prevAsk (*ib_insync.ticker.Ticker* attribute), 71
 prevAskSize (*ib_insync.ticker.Ticker* attribute), 71
 prevBid (*ib_insync.ticker.Ticker* attribute), 71
 prevBidSize (*ib_insync.ticker.Ticker* attribute), 71
 prevLast (*ib_insync.ticker.Ticker* attribute), 71
 prevLastSize (*ib_insync.ticker.Ticker* attribute), 71
 price (*ib_insync.contract.DeltaNeutralContract* attribute), 65
 price (*ib_insync.objects.DOMLevel* property), 90
 price (*ib_insync.objects.Execution* attribute), 78
 price (*ib_insync.objects.HistogramData* attribute), 83
 price (*ib_insync.objects.HistoricalTick* property), 89
 price (*ib_insync.objects.HistoricalTickLast* property), 89
 price (*ib_insync.objects.MktDepthData* property), 90
 price (*ib_insync.objects.TickByTickAllLast* property), 89
 price (*ib_insync.objects.TickData* property), 88
 price (*ib_insync.order.OrderComboLeg* attribute), 46
 price (*ib_insync.order.PriceCondition* attribute), 49

- priceAsk (*ib_insync.objects.HistoricalTickBidAsk* property), 89
- priceBid (*ib_insync.objects.HistoricalTickBidAsk* property), 89
- PriceCondition (class in *ib_insync.order*), 49
- PriceIncrement (class in *ib_insync.objects*), 90
- priceMagnifier (*ib_insync.contract.ContractDetails* attribute), 66
- primaryExchange (*ib_insync.contract.Contract* attribute), 55
- probeContract (*ib_insync.ibcontroller.Watchdog* attribute), 104
- probeTimeout (*ib_insync.ibcontroller.Watchdog* attribute), 104
- projection (*ib_insync.contract.ScanData* attribute), 69
- providerCode (*ib_insync.objects.HistoricalNews* property), 92
- providerCode (*ib_insync.objects.NewsTick* property), 92
- putable (*ib_insync.contract.ContractDetails* attribute), 67
- putOpenInterest (*ib_insync.ticker.Ticker* attribute), 72
- putVolume (*ib_insync.ticker.Ticker* attribute), 72
- pvDividend (*ib_insync.objects.OptionComputation* property), 91
- ## Q
- qualifyContracts() (*ib_insync.ib.IB* method), 14
- qualifyContractsAsync() (*ib_insync.ib.IB* method), 27
- queryDisplayGroups() (*ib_insync.client.Client* method), 33
- ## R
- RaiseRequestErrors (*ib_insync.ib.IB* attribute), 9
- randomizePrice (*ib_insync.order.Order* attribute), 38
- randomizeSize (*ib_insync.order.Order* attribute), 38
- rank (*ib_insync.contract.ScanData* attribute), 69
- ratings (*ib_insync.contract.ContractDetails* attribute), 67
- ratio (*ib_insync.contract.ComboLeg* attribute), 65
- readonly (*ib_insync.ibcontroller.Watchdog* attribute), 103
- realExpirationDate (*ib_insync.contract.ContractDetails* attribute), 67
- realizedPNL (*ib_insync.objects.CommissionReport* attribute), 79
- realizedPnL (*ib_insync.objects.PnL* attribute), 85
- realizedPnL (*ib_insync.objects.PnLSingle* attribute), 86
- realizedPNL (*ib_insync.objects.PortfolioItem* property), 91
- RealTimeBar (class in *ib_insync.objects*), 81
- RealTimeBarList (class in *ib_insync.objects*), 94
- realtimeBars() (*ib_insync.ib.IB* method), 13
- realTimeBarsOptions (*ib_insync.objects.RealTimeBarList* attribute), 94
- refDate (*ib_insync.objects.HistoricalSession* attribute), 87
- 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), 73
- remaining (*ib_insync.order.OrderStatus* attribute), 44
- remaining() (*ib_insync.order.Trade* method), 47
- replaceFAC() (*ib_insync.client.Client* method), 32
- replaceFAC() (*ib_insync.ib.IB* method), 27
- reqAccountSummary() (*ib_insync.client.Client* method), 33
- reqAccountSummary() (*ib_insync.ib.IB* method), 15
- reqAccountSummaryAsync() (*ib_insync.ib.IB* method), 27
- reqAccountUpdates() (*ib_insync.client.Client* method), 32
- reqAccountUpdates() (*ib_insync.ib.IB* method), 15
- reqAccountUpdatesAsync() (*ib_insync.ib.IB* method), 27
- reqAccountUpdatesMulti() (*ib_insync.client.Client* method), 34
- reqAccountUpdatesMulti() (*ib_insync.ib.IB* method), 15
- reqAccountUpdatesMultiAsync() (*ib_insync.ib.IB* method), 27
- reqAllOpenOrders() (*ib_insync.client.Client* method), 32
- reqAllOpenOrders() (*ib_insync.ib.IB* method), 16
- reqAllOpenOrdersAsync() (*ib_insync.ib.IB* method), 28
- reqAutoOpenOrders() (*ib_insync.client.Client* method), 32
- reqAutoOpenOrders() (*ib_insync.ib.IB* method), 16
- reqCompletedOrders() (*ib_insync.client.Client* method), 34
- reqCompletedOrders() (*ib_insync.ib.IB* method), 16
- reqCompletedOrdersAsync() (*ib_insync.ib.IB* method), 28
- reqContractDetails() (*ib_insync.client.Client* method), 32
- reqContractDetails() (*ib_insync.ib.IB* method), 17
- reqContractDetailsAsync() (*ib_insync.ib.IB* method), 28

reqCurrentTime() (*ib_insync.client.Client method*), 33
 reqCurrentTime() (*ib_insync.ib.IB method*), 15
 reqCurrentTimeAsync() (*ib_insync.ib.IB method*), 27
 reqExecutions() (*ib_insync.client.Client method*), 32
 reqExecutions() (*ib_insync.ib.IB method*), 16
 reqExecutionsAsync() (*ib_insync.ib.IB method*), 28
 reqFamilyCodes() (*ib_insync.client.Client method*), 34
 reqFundamentalData() (*ib_insync.client.Client method*), 33
 reqFundamentalData() (*ib_insync.ib.IB method*), 23
 reqFundamentalDataAsync() (*ib_insync.ib.IB method*), 29
 reqGlobalCancel() (*ib_insync.client.Client method*), 33
 reqGlobalCancel() (*ib_insync.ib.IB method*), 15
 reqHeadTimeStamp() (*ib_insync.client.Client method*), 34
 reqHeadTimeStamp() (*ib_insync.ib.IB method*), 21
 reqHeadTimeStampAsync() (*ib_insync.ib.IB method*), 28
 reqHistogramData() (*ib_insync.client.Client method*), 34
 reqHistogramData() (*ib_insync.ib.IB method*), 23
 reqHistogramDataAsync() (*ib_insync.ib.IB method*), 29
 reqHistoricalData() (*ib_insync.client.Client method*), 32
 reqHistoricalData() (*ib_insync.ib.IB method*), 19
 reqHistoricalDataAsync() (*ib_insync.ib.IB method*), 28
 reqHistoricalNews() (*ib_insync.client.Client method*), 34
 reqHistoricalNews() (*ib_insync.ib.IB method*), 26
 reqHistoricalNewsAsync() (*ib_insync.ib.IB method*), 29
 reqHistoricalSchedule() (*ib_insync.ib.IB method*), 19
 reqHistoricalScheduleAsync() (*ib_insync.ib.IB method*), 28
 reqHistoricalTicks() (*ib_insync.client.Client method*), 34
 reqHistoricalTicks() (*ib_insync.ib.IB method*), 20
 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*), 20
 reqMarketRule() (*ib_insync.client.Client method*), 34
 reqMarketRule() (*ib_insync.ib.IB method*), 18
 reqMarketRuleAsync() (*ib_insync.ib.IB method*), 28
 reqMatchingSymbols() (*ib_insync.client.Client method*), 34
 reqMatchingSymbols() (*ib_insync.ib.IB method*), 18
 reqMatchingSymbolsAsync() (*ib_insync.ib.IB method*), 28
 reqMktData() (*ib_insync.client.Client method*), 32
 reqMktData() (*ib_insync.ib.IB method*), 21
 reqMktDepth() (*ib_insync.client.Client method*), 32
 reqMktDepth() (*ib_insync.ib.IB method*), 22
 reqMktDepthExchanges() (*ib_insync.client.Client method*), 34
 reqMktDepthExchanges() (*ib_insync.ib.IB method*), 22
 reqMktDepthExchangesAsync() (*ib_insync.ib.IB method*), 29
 reqNewsArticle() (*ib_insync.client.Client method*), 34
 reqNewsArticle() (*ib_insync.ib.IB method*), 26
 reqNewsArticleAsync() (*ib_insync.ib.IB method*), 29
 reqNewsBulletins() (*ib_insync.client.Client method*), 32
 reqNewsBulletins() (*ib_insync.ib.IB method*), 26
 reqNewsProviders() (*ib_insync.client.Client method*), 34
 reqNewsProviders() (*ib_insync.ib.IB method*), 26
 reqNewsProvidersAsync() (*ib_insync.ib.IB method*), 29
 reqOpenOrders() (*ib_insync.client.Client method*), 32
 reqOpenOrders() (*ib_insync.ib.IB method*), 16
 reqOpenOrdersAsync() (*ib_insync.ib.IB method*), 28
 reqPnL() (*ib_insync.client.Client method*), 34
 reqPnL() (*ib_insync.ib.IB method*), 17
 reqPnLSingle() (*ib_insync.client.Client method*), 34
 reqPnLSingle() (*ib_insync.ib.IB method*), 17
 reqPositions() (*ib_insync.client.Client method*), 33
 reqPositions() (*ib_insync.ib.IB method*), 16
 reqPositionsAsync() (*ib_insync.ib.IB method*), 28
 reqPositionsMulti() (*ib_insync.client.Client method*), 34
 reqRealTimeBars() (*ib_insync.client.Client method*), 33
 reqRealTimeBars() (*ib_insync.ib.IB method*), 18
 reqScannerData() (*ib_insync.ib.IB method*), 23
 reqScannerDataAsync() (*ib_insync.ib.IB method*), 29
 reqScannerParameters() (*ib_insync.client.Client method*), 33
 reqScannerParameters() (*ib_insync.ib.IB method*), 24
 reqScannerParametersAsync() (*ib_insync.ib.IB method*), 29
 reqScannerSubscription() (*ib_insync.client.Client method*), 33

reqScannerSubscription() (*ib_insync.ib*.*IB* method), 24
 reqSecDefOptParams() (*ib_insync.client*.*Client* method), 34
 reqSecDefOptParams() (*ib_insync.ib*.*IB* method), 25
 reqSecDefOptParamsAsync() (*ib_insync.ib*.*IB* method), 29
 reqSmartComponents() (*ib_insync.client*.*Client* method), 34
 reqSmartComponents() (*ib_insync.ib*.*IB* method), 22
 reqSmartComponentsAsync() (*ib_insync.ib*.*IB* method), 29
 reqSoftDollarTiers() (*ib_insync.client*.*Client* method), 34
 reqTickByTickData() (*ib_insync.client*.*Client* method), 34
 reqTickByTickData() (*ib_insync.ib*.*IB* method), 22
 reqTickers() (*ib_insync.ib*.*IB* method), 13
 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), 26
 requestFAAsync() (*ib_insync.ib*.*IB* method), 29
 RequestsInterval (*ib_insync.client*.*Client* attribute), 31
 RequestTimeout (*ib_insync.ib*.*IB* attribute), 9
 reqUserInfo() (*ib_insync.client*.*Client* method), 35
 reqWshEventData() (*ib_insync.client*.*Client* method), 34
 reqWshMetaData() (*ib_insync.client*.*Client* method), 34
 reset() (*ib_insync.client*.*Client* method), 31
 retryDelay (*ib_insync.ibcontroller*.*Watchdog* attribute), 103
 right (*ib_insync.contract*.*Contract* attribute), 54
 routeMarketableToBbo (*ib_insync.order*.*Order* attribute), 41
 rtHistVolatility (*ib_insync.ticker*.*Ticker* attribute), 72
 rtTime (*ib_insync.ticker*.*Ticker* attribute), 72
 rtTradeVolume (*ib_insync.ticker*.*Ticker* attribute), 72
 rtVolume (*ib_insync.ticker*.*Ticker* attribute), 72
 rule80A (*ib_insync.order*.*Order* attribute), 37
 run() (*ib_insync.client*.*Client* method), 31
 run() (*ib_insync.ib*.*IB* static method), 9
 run() (in module *ib_insync.util*), 96
 runAsync() (*ib_insync.ibcontroller*.*Watchdog* method), 104

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), 76
 ScanData (class in *ib_insync.contract*), 69
 ScanDataList (class in *ib_insync.objects*), 94
 scannerSettingPairs (*ib_insync.objects*.*ScannerSubscription* attribute), 77
 ScannerSubscription (class in *ib_insync.objects*), 76
 scannerSubscriptionFilterOptions (*ib_insync.objects*.*ScanDataList* attribute), 94
 scannerSubscriptionOptions (*ib_insync.objects*.*ScanDataList* attribute), 94
 schedule() (*ib_insync.ib*.*IB* static method), 10
 schedule() (in module *ib_insync.util*), 96
 secId (*ib_insync.contract*.*Contract* attribute), 55
 secIdList (*ib_insync.contract*.*ContractDetails* attribute), 67
 secIdType (*ib_insync.contract*.*Contract* attribute), 55
 secType (*ib_insync.contract*.*Contract* attribute), 54
 secType (*ib_insync.objects*.*DepthMktDataDescription* attribute), 84
 secType (*ib_insync.objects*.*ExecutionFilter* attribute), 80
 secType (*ib_insync.order*.*ExecutionCondition* attribute), 51
 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), 88
 setConnectOptions() (*ib_insync.client*.*Client* method), 31
 setServerLogLevel() (*ib_insync.client*.*Client* method), 32

- setTimeout() (*ib_insync.ib.IB* method), 11
 settlingFirm (*ib_insync.order.Order* attribute), 39
 shareholder (*ib_insync.order.Order* attribute), 41
 shares (*ib_insync.objects.Execution* attribute), 78
 shortableShares (*ib_insync.ticker.Ticker* attribute), 72
 shortSaleSlot (*ib_insync.contract.ComboLeg* attribute), 65
 shortSaleSlot (*ib_insync.order.Order* attribute), 38
 side (*ib_insync.objects.Execution* attribute), 78
 side (*ib_insync.objects.ExecutionFilter* attribute), 80
 side (*ib_insync.objects.MktDepthData* property), 90
 size (*ib_insync.objects.DOMLevel* property), 90
 size (*ib_insync.objects.HistoricalTick* property), 89
 size (*ib_insync.objects.HistoricalTickLast* property), 89
 size (*ib_insync.objects.MktDepthData* property), 90
 size (*ib_insync.objects.TickByTickAllLast* property), 89
 size (*ib_insync.objects.TickData* property), 88
 sizeAsk (*ib_insync.objects.HistoricalTickBidAsk* property), 89
 sizeBid (*ib_insync.objects.HistoricalTickBidAsk* property), 89
 sizeIncrement (*ib_insync.contract.ContractDetails* attribute), 67
 sleep() (*ib_insync.ib.IB* static method), 10
 sleep() (in module *ib_insync.util*), 96
 smartComboRoutingParams (*ib_insync.order.LimitOrder* attribute), 42
 smartComboRoutingParams (*ib_insync.order.MarketOrder* attribute), 42
 smartComboRoutingParams (*ib_insync.order.Order* attribute), 39
 smartComboRoutingParams (*ib_insync.order.StopLimitOrder* attribute), 44
 smartComboRoutingParams (*ib_insync.order.StopOrder* attribute), 43
 SmartComponent (class in *ib_insync.objects*), 93
 snapshotPermissions (*ib_insync.ticker.Ticker* attribute), 73
 SoftDollarTier (class in *ib_insync.objects*), 77
 softDollarTier (*ib_insync.order.LimitOrder* attribute), 42
 softDollarTier (*ib_insync.order.MarketOrder* attribute), 42
 softDollarTier (*ib_insync.order.Order* attribute), 40
 softDollarTier (*ib_insync.order.StopLimitOrder* attribute), 44
 softDollarTier (*ib_insync.order.StopOrder* attribute), 43
 solicited (*ib_insync.order.Order* attribute), 40
 specialConditions (*ib_insync.objects.HistoricalTickLast* property), 89
 specialConditions (*ib_insync.objects.TickByTickAllLast* property), 89
 property), 89
 spRatingAbove (*ib_insync.objects.ScannerSubscription* attribute), 76
 spRatingBelow (*ib_insync.objects.ScannerSubscription* attribute), 76
 start() (*ib_insync.ibcontroller.IBC* method), 100
 start() (*ib_insync.ibcontroller.IBController* method), 101
 start() (*ib_insync.ibcontroller.Watchdog* method), 104
 startApi() (*ib_insync.client.Client* method), 33
 startAsync() (*ib_insync.ibcontroller.IBC* method), 100
 startAsync() (*ib_insync.ibcontroller.IBController* method), 101
 startDateTime (*ib_insync.objects.HistoricalSchedule* attribute), 88
 startDateTime (*ib_insync.objects.HistoricalSession* attribute), 87
 startingPrice (*ib_insync.order.Order* attribute), 38
 startLoop() (in module *ib_insync.util*), 97
 startTime (*ib_insync.objects.ConnectionStats* property), 93
 status (*ib_insync.objects.TradeLogEntry* attribute), 86
 status (*ib_insync.order.OrderState* attribute), 45
 status (*ib_insync.order.OrderStatus* attribute), 44
 Stock (class in *ib_insync.contract*), 56
 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), 67
 stockTypeFilter (*ib_insync.objects.ScannerSubscription* attribute), 77
 stop() (*ib_insync.ibcontroller.IBController* method), 101
 stop() (*ib_insync.ibcontroller.Watchdog* method), 104
 stopAsync() (*ib_insync.ibcontroller.IBController* method), 101
 StopLimitOrder (class in *ib_insync.order*), 43
 stopLoss (*ib_insync.order.BasketOrder* property), 48
 StopOrder (class in *ib_insync.order*), 43
 strike (*ib_insync.contract.Contract* attribute), 54
 strikes (*ib_insync.objects.OptionChain* property), 92
 subcategory (*ib_insync.contract.ContractDetails* attribute), 67
 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), 67
 sweepToFill (*ib_insync.order.Order* attribute), 37
 symbol (*ib_insync.contract.Contract* attribute), 54

- symbol (*ib_insync.objects.ExecutionFilter* attribute), 80
symbol (*ib_insync.order.ExecutionCondition* attribute), 51
- ## T
- tag (*ib_insync.contract.TagValue* property), 64
tag (*ib_insync.objects.AccountValue* property), 88
TagValue (class in *ib_insync.contract*), 64
takeProfit (*ib_insync.order.BasketOrder* property), 48
terminate() (*ib_insync.ibcontroller.IBC* method), 100
terminate() (*ib_insync.ibcontroller.IBCController* method), 101
terminateAsync() (*ib_insync.ibcontroller.IBC* method), 100
terminateAsync() (*ib_insync.ibcontroller.IBCController* method), 101
theta (*ib_insync.objects.OptionComputation* property), 91
TickAttrib (class in *ib_insync.objects*), 82
tickAttrib (*ib_insync.objects.OptionComputation* property), 91
TickAttribBidAsk (class in *ib_insync.objects*), 82
tickAttribBidAsk (*ib_insync.objects.HistoricalTickBidAsk* property), 89
tickAttribBidAsk (*ib_insync.objects.TickByTickBidAsk* property), 90
TickAttribLast (class in *ib_insync.objects*), 83
tickAttribLast (*ib_insync.objects.HistoricalTickLast* property), 89
tickAttribLast (*ib_insync.objects.TickByTickAllLast* property), 89
TickBars (class in *ib_insync.ticker*), 75
tickbars() (*ib_insync.ticker.Tickfilter* method), 74
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), 72
TickData (class in *ib_insync.objects*), 88
Ticker (class in *ib_insync.ticker*), 69
ticker() (*ib_insync.ib.IB* method), 13
tickers() (*ib_insync.ib.IB* method), 13
TickerUpdateEvent (class in *ib_insync.ticker*), 74
Tickfilter (class in *ib_insync.ticker*), 74
ticks (*ib_insync.ticker.Ticker* attribute), 72
tickType (*ib_insync.objects.TickByTickAllLast* property), 89
tickType (*ib_insync.objects.TickData* property), 88
tif (*ib_insync.order.Order* attribute), 37
time (*ib_insync.objects.Execution* attribute), 78
time (*ib_insync.objects.ExecutionFilter* attribute), 79
time (*ib_insync.objects.Fill* property), 91
time (*ib_insync.objects.HistoricalNews* property), 92
time (*ib_insync.objects.HistoricalTick* property), 89
time (*ib_insync.objects.HistoricalTickBidAsk* property), 89
time (*ib_insync.objects.HistoricalTickLast* property), 89
time (*ib_insync.objects.MktDepthData* property), 90
time (*ib_insync.objects.RealTimeBar* attribute), 81
time (*ib_insync.objects.TickByTickAllLast* property), 89
time (*ib_insync.objects.TickByTickBidAsk* property), 90
time (*ib_insync.objects.TickByTickMidPoint* property), 90
time (*ib_insync.objects.TickData* property), 88
time (*ib_insync.objects.TradeLogEntry* attribute), 86
time (*ib_insync.order.TimeCondition* attribute), 50
time (*ib_insync.ticker.Bar* attribute), 75
time (*ib_insync.ticker.Ticker* attribute), 70
TimeBars (class in *ib_insync.ticker*), 75
timebars() (*ib_insync.ticker.Tickfilter* method), 74
TimeCondition (class in *ib_insync.order*), 49
timeit (class in *ib_insync.util*), 96
timeRange() (*ib_insync.ib.IB* static method), 10
timeRange() (in module *ib_insync.util*), 96
timeRangeAsync() (*ib_insync.ib.IB* static method), 10
timeRangeAsync() (in module *ib_insync.util*), 97
timeStamp (*ib_insync.objects.NewsTick* property), 92
timeZone (*ib_insync.objects.HistoricalSchedule* attribute), 88
timeZoneId (*ib_insync.contract.ContractDetails* attribute), 67
TimezoneTWS (*ib_insync.ib.IB* attribute), 9
topics() (*ib_insync.flexreport.FlexReport* method), 98
totalQuantity (*ib_insync.order.Order* attribute), 37
Trade (class in *ib_insync.order*), 47
tradeCount (*ib_insync.ticker.Ticker* attribute), 72
TradeLogEntry (class in *ib_insync.objects*), 86
tradeRate (*ib_insync.ticker.Ticker* attribute), 72
trades() (*ib_insync.ib.IB* method), 12
trades() (*ib_insync.ticker.TickerUpdateEvent* method), 74
TRADING_MODE (*ib_insync.ibcontroller.IBCController* attribute), 101
tradingClass (*ib_insync.contract.Contract* attribute), 55
tradingClass (*ib_insync.objects.OptionChain* property), 92
tradingHours (*ib_insync.contract.ContractDetails* attribute), 67
tradingMode (*ib_insync.ibcontroller.IBC* attribute), 99
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), 49
triggerPrice (*ib_insync.order.Order* attribute), 40

- tuple() (*ib_insync.contract.Bag* method), 63
 - tuple() (*ib_insync.contract.Bond* method), 61
 - tuple() (*ib_insync.contract.CFD* method), 60
 - tuple() (*ib_insync.contract.ComboLeg* method), 65
 - tuple() (*ib_insync.contract.Commodity* method), 61
 - tuple() (*ib_insync.contract.ContFuture* method), 58
 - tuple() (*ib_insync.contract.Contract* method), 55
 - tuple() (*ib_insync.contract.ContractDescription* method), 68
 - tuple() (*ib_insync.contract.ContractDetails* method), 68
 - tuple() (*ib_insync.contract.Crypto* method), 64
 - tuple() (*ib_insync.contract.DeltaNeutralContract* method), 66
 - tuple() (*ib_insync.contract.Forex* method), 59
 - tuple() (*ib_insync.contract.Future* method), 57
 - tuple() (*ib_insync.contract.FuturesOption* method), 62
 - tuple() (*ib_insync.contract.Index* method), 59
 - tuple() (*ib_insync.contract.MutualFund* method), 62
 - tuple() (*ib_insync.contract.Option* method), 57
 - tuple() (*ib_insync.contract.ScanData* method), 69
 - tuple() (*ib_insync.contract.Stock* method), 56
 - tuple() (*ib_insync.contract.Warrant* method), 63
 - tuple() (*ib_insync.ibcontroller.IBC* method), 100
 - tuple() (*ib_insync.ibcontroller.IBCController* method), 102
 - tuple() (*ib_insync.ibcontroller.Watchdog* method), 104
 - tuple() (*ib_insync.objects.BarData* method), 81
 - tuple() (*ib_insync.objects.CommissionReport* method), 79
 - tuple() (*ib_insync.objects.DepthMktDataDescription* method), 85
 - tuple() (*ib_insync.objects.Execution* method), 78
 - tuple() (*ib_insync.objects.ExecutionFilter* method), 80
 - tuple() (*ib_insync.objects.HistogramData* method), 84
 - tuple() (*ib_insync.objects.HistoricalSchedule* method), 88
 - tuple() (*ib_insync.objects.HistoricalSession* method), 87
 - tuple() (*ib_insync.objects.NewsProvider* method), 84
 - tuple() (*ib_insync.objects.PnL* method), 85
 - tuple() (*ib_insync.objects.PnLSingle* method), 87
 - tuple() (*ib_insync.objects.RealTimeBar* method), 81
 - tuple() (*ib_insync.objects.ScannerSubscription* method), 77
 - tuple() (*ib_insync.objects.SoftDollarTier* method), 77
 - tuple() (*ib_insync.objects.TickAttrib* method), 82
 - tuple() (*ib_insync.objects.TickAttribBidAsk* method), 83
 - tuple() (*ib_insync.objects.TickAttribLast* method), 83
 - tuple() (*ib_insync.objects.TradeLogEntry* method), 86
 - tuple() (*ib_insync.order.ExecutionCondition* method), 51
 - tuple() (*ib_insync.order.LimitOrder* method), 41
 - 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), 46
 - tuple() (*ib_insync.order.OrderStatus* method), 45
 - 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), 50
 - tuple() (*ib_insync.order.Trade* method), 48
 - tuple() (*ib_insync.order.VolumeCondition* method), 52
 - tuple() (*ib_insync.ticker.Ticker* method), 73
 - TWS_CONFIG_PATH (*ib_insync.ibcontroller.IBCController* attribute), 101
 - TWS_MAJOR_VRSN (*ib_insync.ibcontroller.IBCController* attribute), 101
 - TWS_PATH (*ib_insync.ibcontroller.IBCController* attribute), 101
 - TWSPASSWORD (*ib_insync.ibcontroller.IBCController* attribute), 101
 - twspath (*ib_insync.ibcontroller.IBC* attribute), 99
 - twSettingsPath (*ib_insync.ibcontroller.IBC* attribute), 100
 - TWSUSERID (*ib_insync.ibcontroller.IBCController* attribute), 101
 - twVersion (*ib_insync.ibcontroller.IBC* attribute), 99
- ## U
- underConId (*ib_insync.contract.ContractDetails* attribute), 66
 - underlyingConId (*ib_insync.objects.OptionChain* property), 92
 - underSecType (*ib_insync.contract.ContractDetails* attribute), 67
 - underSymbol (*ib_insync.contract.ContractDetails* attribute), 67
 - undPrice (*ib_insync.objects.OptionComputation* property), 91
 - unrealizedPnL (*ib_insync.objects.PnL* attribute), 85
 - unrealizedPnL (*ib_insync.objects.PnLSingle* attribute), 86
 - unrealizedPNL (*ib_insync.objects.PortfolioItem* property), 91
 - unreported (*ib_insync.objects.TickAttribLast* attribute), 83
 - unsubscribeFromGroupEvents() (*ib_insync.client.Client* method), 33
 - update() (*ib_insync.contract.Bag* method), 64
 - update() (*ib_insync.contract.Bond* method), 61
 - update() (*ib_insync.contract.CFD* method), 60

- update() (*ib_insync.contract.ComboLeg* method), 65
 - update() (*ib_insync.contract.Commodity* method), 61
 - update() (*ib_insync.contract.ContFuture* method), 58
 - update() (*ib_insync.contract.Contract* method), 55
 - update() (*ib_insync.contract.ContractDescription* method), 68
 - update() (*ib_insync.contract.ContractDetails* method), 68
 - update() (*ib_insync.contract.Crypto* method), 64
 - update() (*ib_insync.contract.DeltaNeutralContract* method), 66
 - update() (*ib_insync.contract.Forex* method), 59
 - update() (*ib_insync.contract.Future* method), 58
 - update() (*ib_insync.contract.FuturesOption* method), 62
 - update() (*ib_insync.contract.Index* method), 60
 - update() (*ib_insync.contract.MutualFund* method), 63
 - update() (*ib_insync.contract.Option* method), 57
 - update() (*ib_insync.contract.ScanData* method), 69
 - update() (*ib_insync.contract.Stock* method), 56
 - update() (*ib_insync.contract.Warrant* method), 63
 - update() (*ib_insync.ibcontroller.IBC* method), 100
 - update() (*ib_insync.ibcontroller.IBCController* method), 102
 - update() (*ib_insync.ibcontroller.Watchdog* method), 104
 - update() (*ib_insync.objects.BarData* method), 81
 - update() (*ib_insync.objects.CommissionReport* method), 79
 - update() (*ib_insync.objects.DepthMktDataDescription* method), 85
 - update() (*ib_insync.objects.Execution* method), 79
 - update() (*ib_insync.objects.ExecutionFilter* method), 80
 - update() (*ib_insync.objects.HistogramData* method), 84
 - update() (*ib_insync.objects.HistoricalSchedule* method), 88
 - update() (*ib_insync.objects.HistoricalSession* method), 87
 - update() (*ib_insync.objects.NewsProvider* method), 84
 - update() (*ib_insync.objects.PnL* method), 85
 - update() (*ib_insync.objects.PnLSingle* method), 87
 - update() (*ib_insync.objects.RealTimeBar* method), 82
 - update() (*ib_insync.objects.ScannerSubscription* method), 77
 - update() (*ib_insync.objects.SoftDollarTier* method), 77
 - update() (*ib_insync.objects.TickAttrib* method), 82
 - update() (*ib_insync.objects.TickAttribBidAsk* method), 83
 - update() (*ib_insync.objects.TickAttribLast* method), 83
 - update() (*ib_insync.objects.TradeLogEntry* method), 86
 - update() (*ib_insync.order.ExecutionCondition* method), 51
 - 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), 47
 - update() (*ib_insync.order.OrderCondition* method), 49
 - update() (*ib_insync.order.OrderState* method), 46
 - update() (*ib_insync.order.OrderStatus* method), 45
 - 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), 50
 - update() (*ib_insync.order.Trade* method), 48
 - update() (*ib_insync.order.VolumeCondition* method), 52
 - update() (*ib_insync.ticker.Ticker* method), 73
 - 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), 77
 - validExchanges (*ib_insync.contract.ContractDetails* attribute), 66
 - value (*ib_insync.contract.TagValue* property), 64
 - value (*ib_insync.objects.AccountValue* property), 88
 - value (*ib_insync.objects.PnLSingle* attribute), 86
 - vega (*ib_insync.objects.OptionComputation* property), 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), 81
 - volume (*ib_insync.order.VolumeCondition* attribute), 51
 - volume (*ib_insync.ticker.Bar* attribute), 75
 - volume (*ib_insync.ticker.Ticker* attribute), 71
 - VolumeCondition (class in *ib_insync.order*), 51
 - volumeRate (*ib_insync.ticker.Ticker* attribute), 72
 - vwap (*ib_insync.ticker.Ticker* attribute), 71

W

`waitOnUpdate()` (*ib_insync.ib.IB* method), 10
`waitUntil()` (*ib_insync.ib.IB* static method), 10
`waitUntil()` (in module *ib_insync.util*), 96
`waitUntilAsync()` (in module *ib_insync.util*), 97
`wap` (*ib_insync.objects.RealTimeBar* attribute), 81
`warningText` (*ib_insync.order.OrderState* attribute), 46
`Warrant` (class in *ib_insync.contract*), 63
`Watchdog` (class in *ib_insync.ibcontroller*), 102
`whatIf` (*ib_insync.order.Order* attribute), 39
`whatIfOrder()` (*ib_insync.ib.IB* method), 14
`whatIfOrderAsync()` (*ib_insync.ib.IB* method), 27
`whatToShow` (*ib_insync.objects.BarDataList* attribute),
93
`whatToShow` (*ib_insync.objects.RealTimeBarList* at-
tribute), 94
`whyHeld` (*ib_insync.order.OrderStatus* attribute), 44

Y

`yield_` (*ib_insync.objects.CommissionReport* attribute),
79
`yieldRedemptionDate`
(*ib_insync.objects.CommissionReport* at-
tribute), 79