Module **optibook.synchronous_client**

Classes

```
class Exchange (host: str = None, info_port: int = None, exec_port: int = None,
                username: str = None, password: str = None, admin_password: str = None,
                full_message_logging: bool = False, max_nr_trade_history: int = 100)
```

Initiate an exchange client instance. This is the class you should use to interact with the exchange, i.e. send orders or delete orders, get the newest trades, etc.

Parameters

`host` : `str`
The network location the Exchange Server runs on.

`info_port` : `int`
The port of the Info interface exposed by the Exchange.

`exec_port` : `int`
The port of the Execution interface exposed by the Exchange.

`username` : `str`
Your username.

`password` : `str`
Your password.

`admin_password` : `str`
Reserved for dedicated clients only and can be left empty.

`full_message_logging` : `bool`
If set to to True enables logging on VERBOSE level, displaying among others all messages sent to and received from the exchange.

`max_nr_trade_history` : `int`

Keep at most this number of trades per instrument in history. Older trades will be removed automatically

Methods

```
def amend_order(self, instrument_id: str, *, order_id: int, volume: int)
                -> AmendOrderResponse
```

Amend a specific outstanding limit order on an instrument. E.g. to change its volume.

Parameters

instrument_id : str
The instrument_id of the instrument to delete a limit order for.

order_id : int
The order_id of the limit order to delete.

volume : str
The new volume to change the order to.

Returns

AmendOrderResponse
The AmendOrderResponse returned by the server, containing the success flag as well as error reason should the request have failed. See the doc of that type for more info.

```
def connect(self) -> None
```

Attempt to connect to the exchange. Only a single connection can be made on a single username.

```
def delete_order(self, instrument_id: str, *, order_id: int) -> DeleteOrderResponse
```

Delete a specific outstanding limit order on an instrument.

Parameters

instrument_id : str
The instrument_id of the instrument to delete a limit order for.

order_id : int

The order_id of the limit order to delete.

Returns

`DeleteOrderResponse`

The DeleteOrderResponse returned by the server, containing the success flag as well as error reason should the request have failed. See the doc of that type for more info.

def delete_orders(self, instrument_id: str) -> None

Delete all outstanding orders on an instrument.

Parameters

instrument_id : `str`

The instrument_id of the instrument to delete the orders for.

def disconnect(self) -> None

Disconnect from the exchange.

def get_cash(self) -> float

Get your total cash position.

Returns

`float`

Returns total cash position of the client arising from all cash exchanged on previous buy and sell trades in all instruments.

def get_instruments(self) -> Dict[str, Instrument]

Returns all existing instruments on the exchange

Returns

`typing.Dict[str, Instrument]`

Dict of instrument_id to the instrument definition.

def get_last_price_book(self, instrument_id: str) -> PriceBook

Returns the last received limit order book state for an instrument.

Parameters

instrument_id : str
The instrument_id of the instrument to obtain the limit order book for.

Returns

PriceBook
Returns the last received limit order book state for an instrument.

def get_outstanding_orders(self, instrument_id: str) -> Dict[int, OrderStatus]

Returns the client's currently outstanding limit orders on an instrument.

Parameters

instrument_id : str
The instrument_id of the instrument to obtain outstanding orders for.

Returns

typing.Dict[int, OrderStatus]
Dictionary mapping order_id to OrderStatus objects representing the client's currently outstanding limit orders on an instrument.

def get_pnl(self, valuations: Dict[str, float] = None) -> float

Calculates PnL based on current instrument and cash positions.

For any non-zero position: If the valuations dictionary is provided, uses the valuation provided. If no instrument valuation is provided, falls back on the price of the last public tradetick. If valuation is not provided and no public tradetick is available, no PnL can be calculated.

Parameters

valuations : typing.Dict[str, float]
Optional, dictionary mapping instrument_id to current instrument valuation.

Returns

`float`
: Your current PnL, valued at the last-traded price if no valuations are provided.

`def get_positions(self) -> Dict[str, int]`

Get your current positions.

Returns

`typing.Dict[str, int]`
: Returns a dictionary mapping instrument_id to the current position in the instrument, expressed in amount of lots held.

`def get_positions_and_cash(self) -> Dict[str, Dict[~KT, ~VT]]`

Get your current positions and cash.

Returns

`typing.Dict[str, typing.Dict]`
: Returns a dictionary mapping instrument_id to dictionary of 'volume' and 'cash'. The volume is the current amount of lots held in the instrument and the cash is the current cash position arising from previous buy and sell trades in the instrument.

`def get_trade_history(self, instrument_id: str) -> List[Trade]`

Returns all private trades received for an instrument since the start of this Exchange Client (but capped by max_nr_total_trades). If the total number of trades per instrument is larger than max_nr_total_trades, older trades will not be returned by this function.

Parameters

`instrument_id` : `str`
: The instrument_id of the instrument to obtain the private trade history for.

Returns

`typing.List[Trade]`
: Returns all private trades received for an instrument since the start of this Exchange Client (but capped by max_nr_total_trades).

```
def get_trade_tick_history(self, instrument_id: str) -> List[TradeTick]
```

Returns all public trade ticks received for an instrument since the start of this Exchange Client (but capped by max_nr_total_trades). If the total number of trades per instrument is larger than max_nr_total_trades, older trades will not be returned by this function.

Parameters

instrument_id : `str`
The instrument_id of the instrument to obtain the trade tick history for.

Returns

`typing.List[TradeTick]`
Returns all public trade ticks received for an instrument since the start of this Exchange Client (but capped by max_nr_total_trades).

```
def insert_order(self, instrument_id: str, *, price: float, volume: int, side: str,
                 order_type: str = 'limit') -> InsertOrderResponse
```

Insert a limit or IOC order on an instrument.

Parameters

instrument_id : `str`
The instrument_id of the instrument to insert the order on.

price : `float`
The (limit) price of the order.

volume : `int`
The number of lots in the order.

side : `str`
'bid' or 'ask', a bid order is an order to buy while an ask order is an order to sell.

order_type : `str`
'limit' or 'ioc', limit orders stay in the book while any remaining volume of an IOC that is not immediately matched is cancelled.

Returns

`InsertOrderResponse`
The InsertOrderResponse returned by the server, containing the success flag as well as an order id and an error reason should the request have failed. See the doc of that type

for more info.

def is_connected(self) -> bool

Tells you if the client is currently connected to the exchange.

Returns

`bool`
True if you are connected, otherwise false.

def poll_new_trade_ticks(self, instrument_id: str) -> List[TradeTick]

Returns the public trades received for an instrument since the last time this function was called for that instrument. Public trades are trades between two other parties, in which you may or may not have been involved.

Parameters

`instrument_id` : `str`
The instrument_id of the instrument to poll the trade ticks for.

Returns

`typing.List[TradeTick]`
Returns the public trades received for an instrument since the last time this function was called for that instrument.

def poll_new_trades(self, instrument_id: str) -> List[Trade]

Returns the private trades received for an instrument since the last time this function was called for that instrument.

Parameters

`instrument_id` : `str`
The instrument_id of the instrument to poll the private trades for.

Returns

`typing.List[Trade]`

Returns the private trades received for an instrument since the last time this function was called for that instrument.

Index (docs?page=optibook)

Classes

Exchange

amend_order
connect
delete_order
delete_orders
disconnect
get_cash
get_instruments
get_last_price_book
get_outstanding_orders
get_pnl
get_positions
get_positions_and_cash
get_trade_history
get_trade_tick_history
insert_order
is_connected
poll_new_trade_ticks
poll_new_trades