hummingbot

Crypto Trading Bots

Scripts Strategies Cheat Sheet

> Getting started

Last edit: Dec, 2022
Author: Federico Cardoso @dardonacci

1. Install Anaconda
2. Open a terminal and run:
```
>>> git clone https://github.com/hummingbot/hummingbot.git
>>> cd hummingbot
>>> ./install
>>> conda activate hummingbot
>>> ./compile
```
3. Code your script under the `scripts` folder!

> Scripts basics

Configuration

- The Scripts are a subclass of **ScriptStrategyBase**
- You can define the variables that you will use as class variables, there is no configuration file for scripts.

Markets

- Define the connectors and trading pairs, in the class variable **markets**, with the following structure:
  - `Dict["connector_name", Set(Trading pairs)]`

Execution

- The method on_tick is executed every **tick_size**.
- The **tick_size** by default is 1 second.

> Market operations

Create and Cancel orders

- `self.buy(connector_name, trading_pair, amount, order_type, price, position_action)`
- `self.sell(connector_name, trading_pair, amount, order_type, price, position_action)`
- `self.cancel(connector_name, trading_pair, order_id)`

Note: position_action is only used in perpetuals.

> Account data

Balance

- `self.get_balance_df()`
  - Returns a **DataFrame** with the following columns: `["Exchange", "Asset", "Total Balance", "Available Balance"]`

Open Orders

- `self.active_orders_df()`
  - Returns a **DataFrame** with the following columns: `["Exchange", "Market", "Side", "Price", "Amount", "Age"]`

> Events

To handle different market events in the strategy by implementing the following methods.

- `did_create_buy_order(self, event: BuyOrderCreatedEvent)`
- `did_create_sell_order(self, event: SellOrderCreatedEvent)`
- `did_fill_order(self, event: OrderFilledEvent)`
- `did_fail_order(self, event: MarketOrderFailureEvent)`
- `did_cancel_order(self, event: OrderCancelledEvent)`
- `did_expire_order(self, event: OrderExpiredEvent)`
- `did_complete_buy_order(self, event: BuyOrderCompletedEvent)`
- `did_complete_sell_order(self, event: SellOrderCompletedEvent)`

> Other

Rate Oracle

- Provides conversion rates for any given pair token symbols in both async and sync fashions.
- Sync method: `RateOracle.get_instance().get_pair_rate(trading_pair)`
- Async method: `RateOracle.get_instance().rate_async(trading_pair)`

Notifiers

To send notifications to the Hummingbot Application using the following methods:

- `self.notify_hb_app(msg)`
- `self.notify_hb_app_with_timestamp(msg)`

Note: if you have the Telegram integration activated, you will receive the notifications there too.

Status

- When you run the `status` command in the app, you will receive the information that is coded under the method `format_status`.
- You can implement this method in your script to show the info that you want
- By default, the format status shows the balances and active orders. (check the implementation in `ScriptStrategyBase`)

> Connectors

Accessing the connectors

- They are stored in the instance variable **connectors** with the following structure:
  - `Dict["connector_name", ConnectorBase]`
    - e.g.`self.connectors["binance"]` will return the Binance exchange class.

Connectors Methods

- Best ask: **connector**.`get_price(trading_pair, is_buy: True)`
- Best bid: **connector**.`get_price(trading_pair, is_buy: False)`
- Mid-price: **connector**.`get_mid_price(trading_pair)`
- Order book: **connector**.`get_order_book(trading_pair)`
  - Returns a `CompositeOrderBook` and the most common methods are:
    - `ask_entries()` --> Iterator of OrderBookRow
    - `bid_entries()` --> Iterator of OrderBookRow
    - `snapshot()` --> Tuple(Bids as DataFrame, Asks as DataFrame)

Example:
- `self.connectors["binance"].get_mid_price("ETH-USDT")`

Querying the Order Book

Use these methods to compute metrics efficiently:

- **connector**.`get_vwap_for_volume(trading_pair, is_buy, volume)`
- **connector**.`get_price_for_volume(trading_pair, is_buy, volume)`
- **connector**.`get_quote_volume_for_base_amount(trading_pair, is_buy, base_amount)`
- **connector**.`get_volume_for_price(trading_pair, is_buy, price)`
- **connector**.`get_quote_volume_for_price(trading_pair, is_buy, price)`

Returns a `ClientOrderBookQueryResult` class with:
  - `query_price`
  - `query_volume`
  - `result_price`
  - `result_volume`

> Accounting

Order Candidate

- `OrderCandidate(trading_pair, is_maker, order_type, order_side, amount, price)`
- Has methods to populate the object with the collateral needed, the fees, and potential returns.

Budget Checker

- **connector**.`budget_checker.adjust_candidate(OrderCandidate, all_or_none=True)`
- **connector**.`budget_checker.adjust_candidates(List[OrderCandidate], all_or_none=True)`

Note: This checks if the balance is enough to place the order, `all_or_none=True` will set the amount to 0 on insufficient balance and `all_or_none=False` will adjust the order size to the available balance.