mql5$_{z}mq_{b}acktrader Documentation$

Release 0.1.0

R. Martin Parrondo

Mar 02, 2020

Contents:

CHAPTER 1

mql5_zmq_backtrader

Project developed to work as a server for Python trading community. It is based on ZeroMQ sockets and uses JSON format to communicate messages. It is a python library for the ZeroMQ API within backtrader framework. It allows rapid trading algo development. For details of API behavior, please see the online API document.

- Free software: MIT license
- Documentation: https://mql5-zmq-backtrader.readthedocs.io.

1.1 Features

- TODO

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install mql5_zmq_backtrader, run this command in your terminal:

```
$ pip install mql5_zmq_backtrader
```

This is the preferred method to install mql5_zmq_backtrader, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for mql5_zmq_backtrader can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/parrondo/mql5_zmq_backtrader
```

Or download the tarball:

```
$ curl -OJL https://github.com/parrondo/mql5_zmq_backtrader/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use mql5_zmq_backtrader in a project:

```
import mql5_zmq_backtrader
```

CHAPTER 4

mql5_zmq_backtrader

4.1 mql5_zmq_backtrader package

4.1.1 Submodules

4.1.2 mql5_zmq_backtrader.adapter module

class mql5_zmq_backtrader.adapter.**Adapter**(*raw*)
Bases: object

class mql5_zmq_backtrader.adapter.**BalanceAdapter**(*raw*)
Bases: *mql5_zmq_backtrader.adapter.Adapter*

class mql5_zmq_backtrader.adapter.**OrderAdapter**(*raw*)
Bases: *mql5_zmq_backtrader.adapter.Adapter*

class mql5_zmq_backtrader.adapter.**PositionAdapter**(*raw*)
Bases: *mql5_zmq_backtrader.adapter.Adapter*

4.1.3 mql5_zmq_backtrader.cli module

Console script for mql5_zmq_backtrader.

4.1.4 mql5_zmq_backtrader.mql5_zmq_backtrader module

Main module.

4.1.5 mql5_zmq_backtrader.mt5broker module

class mql5_zmq_backtrader.mt5broker.**MTraderBroker**(***kwargs*)
Bases: backtrader.broker.BrokerBase

Broker implementation for MetaTrader 5.

This class maps the orders/positions from MetaTrader to the internal API of *backtrader*.

Params:

- *use_positions* (Ram default:*True*): When connecting to the broker provider use the existing positions to kickstart the broker.

  Set to *False* during instantiation to disregard any existing position

MTraderBroker(*datas=None*)

buy(*owner*, *data*, *size*, *price=None*, *plimit=None*, *exectype=None*, *valid=None*, *tradeid=0*, *oco=None*, *trailamount=None*, *trailpercent=None*, *parent=None*, *transmit=True*, ***kwargs*)

cancel(*order*)

data_started(*data*)

frompackages = ()

get_notification()

getcash()

getposition(*data*, *clone=True*)

getvalue(*datas=None*)

next()

notify(*order*)

orderstatus(*order*)

packages = ()

params
alias of `backtrader.metabase.AutoInfoClass_BrokerBase_MTraderBroker`

sell(*owner*, *data*, *size*, *price=None*, *plimit=None*, *exectype=None*, *valid=None*, *tradeid=0*, *oco=None*, *trailamount=None*, *trailpercent=None*, *parent=None*, *transmit=True*, ***kwargs*)

start()

stop()

class mql5_zmq_backtrader.mt5broker.**MTraderCommInfo**
Bases: `backtrader.comminfo.CommInfoBase`

frompackages = ()

getoperationcost(*size*, *price*)
Returns the needed amount of cash an operation would cost

getvaluesize(*size*, *price*)
Returns the value of size for given a price. For future-like objects it is fixed at size * margin

packages = ()

params
alias of `backtrader.metabase.AutoInfoClass_CommInfoBase_MTraderCommInfo`

class mql5_zmq_backtrader.mt5broker.**MetaMTraderBroker**(*name*, *bases*, *dct*)
Bases: `backtrader.broker.MetaBroker`

4.1.6 mql5_zmq_backtrader.mt5data module

class mql5_zmq_backtrader.mt5data.**MTraderData**(***kwargs*)

Bases: backtrader.feed.DataBase

MTrader Data Feed.

TODO: implement tick data. Main problem is that Backtrader is not tick oriented. TODO: test backfill_from

Params:

- *historical* (default: *False*)

  If set to *True* the data feed will stop after doing the first download of data.

  The standard data feed parameters *fromdate* and *todate* will be used as reference.

- *backfill* (default: *True*)

  Perform backfilling after a disconnection/reconnection cycle. The gap duration will be used to download the smallest possible amount of data

- *backfill_from* (default: *None*)

  An additional data source can be passed to do an initial layer of backfilling. Once the data source is depleted and if requested, backfilling from IB will take place. This is ideally meant to backfill from already stored sources like a file on disk, but not limited to.

- *include_last* (default: *False*)

  Last historical candle is not closed. It will be updated in live stream

- *reconnect* (default: *True*)

  Reconnect when network connection is down

alias = ()

aliased = ''

frompackages = ()

haslivedata()

islive()

True notifies *Cerebro* that *preloading* and *runonce* should be deactivated

linealias

alias of backtrader.metabase.AutoInfoClass_la_LineSeries_la_DataSeries_la_OHLC_la_OHLCD

lines

alias of backtrader.lineseries.Lines_LineSeries_DataSeries_OHLC_OHLCDateTime_AbstractDa

packages = ()

params

alias of backtrader.metabase.AutoInfoClass_LineRoot_LineMultiple_LineSeries_DataSeries_

plotinfo

alias of backtrader.metabase.AutoInfoClass_pi_LineSeries_pi_DataSeries_pi_OHLC_pi_OHLCD

plotlines

alias of backtrader.metabase.AutoInfoClass_pl_LineSeries_pl_DataSeries_pl_OHLC_pl_OHLCD

setenvironment(*env*)

Receives an environment (cerebro) and passes it over to the store it belongs to

start()
Starts the MTrader connection and gets the real contract and contractdetails if it exists

stop()
Stops and tells the store to stop

class mql5_zmq_backtrader.mt5data.**MetaMTraderData**(*name*, *bases*, *dct*)
Bases: backtrader.feed.MetaAbstractDataBase

4.1.7 mql5_zmq_backtrader.mt5store module

class mql5_zmq_backtrader.mt5store.**MTraderAPI**(*host=None*)
Bases: object

This class implements Python side for MQL5 JSON API See https://github.com/khramkov/MQL5-JSON-API for docs

construct_and_send(***kwargs*) → dict
Construct a request dictionary from default and send it to server

live_socket(*context=None*)
Connect to socket in a ZMQ context

streaming_socket(*context=None*)
Connect to socket in a ZMQ context

exception mql5_zmq_backtrader.mt5store.**MTraderError**(**args*, ***kwargs*)
Bases: Exception

class mql5_zmq_backtrader.mt5store.**MTraderStore**(*host='localhost'*)
Bases: object

Singleton class wrapping to control the connections to MetaTrader.

Balance update occurs at the beginning and after each transaction registered by '_t_streaming_events'.

BrokerCls
alias of *mql5_zmq_backtrader.mt5broker.MTraderBroker*

DataCls
alias of *mql5_zmq_backtrader.mt5data.MTraderData*

broker_threads()

cancel_order(*oid*, *symbol*)

candles(*dataname*, *dtbegin*, *dtend*, *timeframe*, *compression*, *include_first=False*)

check_account() → None
Get MetaTrader 5 account settings

close_position(*oid*, *symbol*)

frompackages = ()

get_balance()

get_cash()

get_granularity(*timeframe*, *compression*)

get_notifications()
Return the pending "store" notifications

get_positions()

get_value()

classmethod getbroker(**args*, ***kwargs*)
Returns broker with *args, **kwargs from registered *BrokerCls*

classmethod getdata(**args*, ***kwargs*)
Returns *DataCls* with args, kwargs

order_cancel(*order*)

order_create(*order*, *stopside=None*, *takeside=None*, ***kwargs*)
Creates an order

packages = ()

params
alias of `backtrader.metabase.AutoInfoClass_MTraderStore`

put_notification(*msg*, **args*, ***kwargs*)

start(*data=None*, *broker=None*)

stop()

streaming_events()

class mql5_zmq_backtrader.mt5store.**MetaSingleton**(*name*, *bases*, *dct*)
Bases: `backtrader.metabase.MetaParams`

Metaclass to make a metaclassed class a singleton

exception mql5_zmq_backtrader.mt5store.**ServerConfigError**(**args*, ***kwargs*)
Bases: *mql5_zmq_backtrader.mt5store.MTraderError*

exception mql5_zmq_backtrader.mt5store.**ServerDataError**(**args*, ***kwargs*)
Bases: *mql5_zmq_backtrader.mt5store.MTraderError*

exception mql5_zmq_backtrader.mt5store.**StreamError**(**args*, ***kwargs*)
Bases: *mql5_zmq_backtrader.mt5store.MTraderError*

exception mql5_zmq_backtrader.mt5store.**TimeFrameError**(**args*, ***kwargs*)
Bases: *mql5_zmq_backtrader.mt5store.MTraderError*

4.1.8 Module contents

Top-level package for mql5_zmq_backtrader. Project developed to work as a server for Python trading community. It is based on ZeroMQ sockets and uses JSON format to communicate messages. It is a python library for the ZeroMQ API within backtrader framework. It allows rapid trading algo development. For details of API behavior, please see the online API document.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/parrondo/mql5_zmq_backtrader/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

mql5_zmq_backtrader could always use more documentation, whether as part of the official mql5_zmq_backtrader docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/parrondo/mql5_zmq_backtrader/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *mql5_zmq_backtrader* for local development.

1. Fork the *mql5_zmq_backtrader* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mql5_zmq_backtrader.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mql5_zmq_backtrader
$ cd mql5_zmq_backtrader/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 mql5_zmq_backtrader tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/parrondo/mql5_zmq_backtrader/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_mql5_zmq_backtrader
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- R. Martin Parrondo

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.0 (2020-02-16)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

H

I

L

M

N

O

P

S

T