PyNinjaTrader_API.

Table of contents

Changes

Date	Version	Changes
10-07-24	V3101	Original version.

Introduction

The PyNinjaTrader_API (DLL) is an interface between C# code and MT4/5 terminals. Via this API the user can f.i. retrieve candles, ticks, instrument info, manage trades and so on. The communication between the API and the MT EA is based on sockets. The MT EA is the server, the C#/API code is the client. It is always ask followed by an answer.The system consists of 2 components:

- An MT4 or MT5 EA. This EA has to run on a MT4 / MT5 terminal in only one chart.
- A C# script/program (coded by the user). This script uses the DLL for the communication with the EA. All comunication is ascii based.

The API can be used in NT8, cTrader and 100% own c# code. Testing is on the way.

Functions.

General

1. The PyNinjaTrader_API is coded as a C# class.
2. After the execution of a function, the *MT.command_OK* property will be set to *True* or *False*.

Time out is set to 60 seconds as default. There is a separate function to change the 'time out' time.

The result of a function can have different formats:

- Bool, true or false
- ```
  /// <summary>
  /// struct used as result for several functions
  /// status: bool: function executed successfull or not
  /// dict: retrieved values, key and value are string variables
  /// </summary>
  public struct resultDict
  {
          public bool status;
          public Dictionary<string, string> dict;
  }
  ```
- ```
  ///<summary>
  /// struct used as result for candle/bar functions
  /// status: bool function executed successfull or not
  /// data: List<Tuple>long, double, double, double, double, double</Tuple>
  /// </summary>
  public struct candles
  {
          public bool status;
          public List<Tuple<long, double, double, double, double, double>> data;
  }
  ```

- /// <summary>
  /// struct for open orders result
  /// status: bool function executed successfull or not
  /// data: List<(long ticket, string instrument, string order_type, long magic_number, double
  /// volume, double open_price, long open_time, double stop_loss, double take_profit,
  /// string comment)>
  /// </summary>
  public struct open_orders
  {
  public bool status;
  public List<(long ticket, string instrument, string order_type, long magic_number,
  double volume, double open_price, long open_time, double stop_loss, double
  take_profit, string comment)> order;
  }

- /// <summary>
  /// struct for deleted orders result
  /// status: bool function executed successfull or not
  /// data: List<(long ticket, string instrument, string order_type,
  /// long magic_number, double volume, double open_price, long open_time,
  /// double stop_loss, double take_profit,
  /// double delete_price, long delete_time, string comment)>
  /// </summary>
  public struct deleted_orders
  {
  public bool status;
  public List<(long ticket, string instrument, string order_type, long magic_number,
  double volume, double open_price, long open_time, double stop_loss,
  double take_profit, double delete_price, long delete_time, string comment)> order;
  }

- /// <summary>
  /// struct for open positions result
  /// status: bool function executed successfull or not
  /// data: List<(long ticket, string instrument, long order_ticket, string position_type, long
  /// magic_number, double volume, double open_price, long open_time, double stop_loss,
  /// double take_profit, string comment, double profit, double swap, double commission)>
  /// </summary>
  public struct open_positions
  {
  public bool status;
  public List<(long ticket, string instrument, long order_ticket, string position_type,
  long magic_number, double volume, double open_price, long open_time,
  double stop_loss, double take_profit, string comment, double profit, double swap,
  double commission)> open_position;
  }

1. Instantiation.

declaration

The DLL has to added to the references and declared in the heading:

using PyNinjaTrader_API;

instantiate

PyNinjaTrader_API.client client = new client();

2. Connect to server(MT terminal)

At connection time a broker instrument dictionary has to be passed as a parameter. This dictionary is a lookup table for translating general instrument / symbol names into specific broker instrument / symbol names.

Instrument lookup dictionary, key=general instrument /symbol name, value=broker instrument / symbol name

brokerInstrumentsLookup = {'EURUSD':'EURUSD.ecn', 'GOLD':'XAUUSD', 'DAX':'GER40'}

connect to server local or to computer in same local network
Connected = client.Connect(ip_server: '127.0.0.1', port_number: =10014,
instrument_lookup: brokerInstrumentsLookup,
authorization: 'Author_123');
or
Connected = client.Connect(server: '192.168.0.103', port: 10014,
instrument_lookup: brokerInstrumentsLookup,
authorization_code: 'Author_123');

'192.168.0.103' = server. In this case other computer in same local network. Of course we can also connect to MT4/5 running on a computer outside local network, like a VPS.

11111 = port (number). Server socket of the MT4/5 EA must use same port.

brokerInstrumentLookup = dictionary<string, string>

authorization_code = 'Author_123'. This parameter can be skipped. The authorization_code'= None as default. This code and the EA authorization_code must match.

Connected = bool, *True* or *False*.
If connection is made the *MT.connected* property will be set to *True*. If no connection *MT.connected* property will be set to *False*.

1. Disconnect from server(MT terminal)

Disconnected = client.Disconnect();

Disconnected = bool, *True* or *False*.

3. Check connection.

CheckAlive = client.Check_connection();

CheckAlive = bool, *True* or *False*. In this case a check will be done if the socket communication is still functioning.

4. Change time out value.

Result = client.Set_timeout(timeout_in_seconds: 120);

120 = time out value in seconds. The MT4/5 EA must answer within this time, if not a socket timeout will be generated.

Result = bool, always *True, this is a DLL setting*

5. Retrieve broker server time.

resultDict = client.Get_broker_server_time();

resultDict = struct with following information:
status: true/false
dict:
date. = 2024-01-12 11:10:15

6. Get static account information.

resultDict = client.Get_static_account_info();

resultDict = struct with following information:
status: true/false
dict:
name=…..
login = 11117869
currency =USD
type = demo
leverage = 100
trade_allowed = True
limit_orders = 200
margin_call = 100.0

margin_close = 50.0
company = . ICM

7. Get dynamic account information.

resultDict = client.Get_dynamic_account_info();

resultDict = struct with the following information:
status: true/false
dict:
balance = 3400.0
equity = 3350.0
profit = -50.0
margin = 40.6
margin_level = 8106.05
margin_free = 3101.64

8. Get instrument information.

resultDict = client.Get_instrument_info(instrument: "EURUSD");

'EURUSD' = instrument.
resultDict = struct with the following information:
status: true/false
dict:
Instrument = EURUSD
digits = 5
max_lotsize = 200.0
min_lotsize = 0.01
lot_step = 0.01
point = 1e-05
tick_size = 1e-05
tick_value = 1.0
stop_level = 0

2. Get last tick information.

resultDict = client.Get_last_tick_info(instrument: "EURUSD")

resultDict = struct with the following information:
status: true/false
dict:
instrument=EURUSD
date=1591401419
ask=1.12907
bid=1.129
last=0.0
volume=123

3. Get actual bar information

resultDict = client.Get_actual_candle_info(instrument: "EURUSD",
timeframe: client.get_timeframe_value("H4"))

client.get_timeframe_value("H4") converts timeframe/period to integer value

resultDict = struct with the following information:
status: true/false
dict:
instrument = EURUSD
date = 1591315200
open = 1.13369
high = 1.13838
low = 1.12784
close = 1.129
volume = 98291

9. Get last x bars from now.

candles = client.Get_last_x_candles_from_now(instrument: "EURUSD",
timeframe: client.get_timeframe_value("M1"), nbr_of_bars: 1000)

candles = struct with the following information:
status: true/false
List<(date, open, high, low, close, volume)>

10. Open order.

open market
NewOrder = client.Open_order(instrument: "EURUSD", order_type: "buy", volume: 0.01, open_price: 0.0, slippage: 10, magic_number: 2000, stop_loss: 0.0, take_profit: 0.0, comment: "Test");

open pending order
NewOrder = client.Open_order(instrument: "EURUSD", order_type: "buy_stop", volume: 0.04, open_price: 1.0870, slippage: 10, magic_number: 2000, stop_loss: 1.0830, take_profit: 1.0950, comment:"Test");

'EURUSD' = instrument.
'buy' = order type ('buy', 'sell', 'buy_stop', 'sell_stop', 'buy_limit', 'sell_limit').
0.02 = volume/lot size.
0.0 = open_price. For market orders price will be zero (0.0), for pending orders price must have an appropriate value.
10 = slippage.
1000 = magic_number.
1.0830 = stop_loss. The stop loss value is a market price (not in delta pips), if 0.0 then no stop_loss set.
1.0950 = take_profit. The take profit is a market price (not in delta pips), if 0.0 then no take_profit set.
Test = comment. The comment may not contain the characters !#$, these are used internally.

NewOrder: integer value with the ticket number of the order. If the value is: -1, then error occurred.
For the type of error check:

- client.command_return_error
- client.order_return_message
- client.order_error

11. Set SL and TP for position.

ModifyPosition = client.Set_sl_and_tp_for_position(ticket: 53136604, stop_loss: 0.0, take_profit: 1.11001);

ModifyPosition = bool, *True* or *False*,
For the type of error check:

- client.command_return_error
- client.order_return_message
- client.order_error

12. Set SL and TP for order (pendings).

ModifyOrder = client.Set_sl_and_tp_for_order(ticket: 53136804, stop_loss: 0.0, take_profit: 1.12001);

ModifyOrder = bool, *True* or *False*
For the type of error check:

- client.command_return_error
- client.order_return_message
- client.order_error

13. Get all (open)orders.

AllOrders = client.Get_all_orders();

AllOrders = struct with the following information:
status: true/false
List< (long ticket, string instrument, string order_type, long magic_number, double volume, double open_price, long open_time, double stop_loss, double take_profit, string comment)>

14. Get all deleted orders within window.

AllDeletedOrders = client.Get_all_deleted_orders_within_window((date_from: datetime(2020, 6, 3, tzinfo=timezone), date_to: datetime.now()));

date_from = datetime(2020, 6, 3, tzinfo=timezone
date_to = datetime.now() + **"delta broker time and local time"**

AllDeletedOrders= struct with the following information:
status: true/false
List<(long ticket, string instrument, string order_type, long magic_number, double volume, double open_price, long open_time, double stop_loss, double take_profit, double delete_price, long delete_time, string comment)>

15. Get all deleted orders.

AllDeletedOrders = client.Get_all_deleted_orders();

AllDeletedOrders= struct with the following information:
status: true/false
List< (long ticket, string instrument, string order_type, long magic_number, double volume, double open_price, long open_time, double stop_loss, double take_profit, double delete_price, long delete_time, string comment)>

16. Get all (open) positions.

AllPositions = client.Get_all_open_positions();

AllPositions = struct with the following information:
status: true/false
List<(long ticket, string instrument, long order_ticket, string position_type, long magic_number, double volume, double open_price, long open_time, double stop_loss, double take_profit, string comment, double profit, double swap, double commission)>

17. Get all closed positions within window.

timezone = pytz.timezone("Etc/UTC")

AllClosedPositions = client.Get_closed_positions_within_window (date_from: datetime(2020, 6, 3, tzinfo=timezone), date_to: datetime.now());

date_from = datetime(2020, 6, 3, tzinfo=timezone)
date_to = datetime.now()

AllClosedPositions = struct with the following information:
status: true/false
List<(long ticket, string instrument, long order_ticket, string position_type, long magic_number, double volume, double open_price, long open_time, double stop_loss, double take_profit, double close_price, long close_time, string comment, double profit, double swap, double commission)>

Be aware:

- The positions must be opened and closed within the window.
- For MT4 the positions must be in the history of the MT4 terminal
- That there is probably a difference between local time and broker server time. Positions are in broker server time.

18. Get all closed positions.

AllClosedPositions = MT.Get_all_closed_positions();

AllClosedPositions = struct with the following information:
status: true/false
List<(long ticket, string instrument, long order_ticket, string position_type, long magic_number, double volume, double open_price, long open_time, double stop_loss, double take_profit, double close_price, long close_time, string comment, double profit, double swap, double commission)>

Be aware:

- For MT4 positions must be in history of the MT4 terminal
- That there is probably a difference between local time and broker server time. Positions are in broker server time.