JsTrader.

Table of contents

Changes

Changes

Date	Version	Changes
17-09-24	V1.01	Original version.

Introduction

The JsTraderApi is an interface between JavaScript 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's is based on sockets. The MT EA is the server, the JsTraderApi 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 JavaScript script/program (coded by the user). This script uses the JsTraderApi for the communication with the EA. All communication is ascii based.

Functions.

General

1. The JsTraderApi is coded as a JavaScript class.
2. After the execution of a function, the *api.command_OK* property will be set to *True* or *False*. So easy to check the result of the action.

Time out is set to 5 seconds as default. There is a separate function to change the 'time out' time.

1. Instantiation.

```
## instantiate

// get the API

const JsTraderApi = require('./jstrader_api_V1_01');                // adjust the path if needed
const JsTraderApiError = require('./jstrader_api_V1_01');          // adjust the path if needed


// Create an instance of the API
const api = new JsTraderApi();
```

The JsTraderApiError class will give all kind of errors if things go wrong.

2. Connect to server.

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.

```
var LOOKUPTABLE = {
   'EURUSD': 'EURUSD',
   'GBPUSD': 'GBPUSD'
}

/**
    * Connects to a MT4 or MT5 EA/Bot.
    *
    * @param {string} [server='127.0.0.1'] - Server IP address.
    * @param {number} [port=1111] - Port number.
    * @param {Object} [instrumentLookup={}] - Dictionary with general instrument names and broker
                                                instrument names. Act as translation table.
    * @param {string} [authorizationCode='None'] - Authorization code for extra security.
    *                                                   Same code must be used in the MT EA
    * @returns {Promise<Object>} - List[]
    *      List[0] - bool: true or false
    *      List[1] – string: server ip address
    * @throws {JsTraderApiError} If connection fails or instrumentLookup list is empty.
    */
```

## connect to server local or to computer in same local network

```
const connected = await api.Connect(server,  port,  LOOKUPTABLE,  'None');
```

or

```
const connected = await api.Connect(server= '192.168.0.103', port = 10014,
                    instrumentLookup = LOOKUPTABLE,
                    authorizationCode = ’Author_123’);
```

Disconnect from server.

```
    /**
        * Closes the socket connection to a MT4 or MT5 EA bot.
        *
        * @throws {JsTraderApiError} If the socket is not initialized or an error occurs during
    disconnection.
        */
const disconnected = await api.Disconnect();
```

3. Check connection.

```
/**
* Checks if connection with MT terminal/EA bot is still active.
*
* @returns {Promise<boolean>} – true = connected,  false = not connected
* @throws {JsTraderApiError} If the connection check fails, f.i socket not initialized.
 */

Const connected  = await api.Check_connection();
```

4. Disconnect from MT terminal.

```
/**
    * Closes the socket connection to a MT4 or MT5 EA bot.
    * @throws {JsTraderApiError} If the socket is not initialized or an error occurs during disconnection.
    */

await api.Disconnect()
```

5. Change time out value.

```
/**
    * Set timeout value for socket communication with MT4 or MT5 EA/Bot.
    *
    * @param {number} [timeoutInSeconds = 5] - The timeout value in seconds.
    * @throws {JsTraderApiError} If the socket is not initialized.
*/
await api.Set_timeout(timeoutInSeconds = 120);
```

6. Get broker server time.

```
/**
    * Retrieve broker server time.
    *
    * @returns {Promise<Object>} List[]
    *      List[0] = status, true or false
    *      List[1] = datetime: Boker time
    * @throws {JsTraderApiError} If the connection check fails.
    */
Const broker_server_time = await api.Get_broker_server_time();
```

7. Get static account information.

```
/**
  *  Retrieve static account information.
  *
  *  @returns {Promise<Object>} List[] - Account information.
  *       List[0] - bool: status, true or false
  *       List[1] - {name, number, currency, type, leverage, trading_allowed, limit_orders, margin_call,
                   margin_close, company }
  *  @throws {JsTraderApiError} If retrieval of account information fails.
  * /
```

```
const staticAccount = await api.Get_static_account_info();
```

8. Get dynamic account information.

```
/**
  * Retrieve dynamic account information.
  * @returns {Promise<Object>} List[] - Account information.
  *       List[0] - bool: status, true or false
  *       List[1] - {balance, equity, profit, margin, margin_level, margin_free }
  * @throws {JsTraderApiError} If retrieval of account information fails.
  * */
```

```
const dynamicAccount = await api.Get_dynamic_account_info();
```

9. Get instrument information.

```
/**
  * Retrieve instrument info/parameters.
  *
  * @param {string} [instrumentName] - Name of instrument.
  * @returns {Promise<Object>} - List[boolean, {}]
  *       List[0] - bool: status, true or false
  *       List[1] – dict: {instrument, digits, max_lotsize, min_lotsize, lot_step, point,
  *                   tick_size, tick_value, swap_long, swap_short, stop_level, contract_size}
  * @throws {JsTraderApiError} If connection fails.
  */
```

```
const instrumentInfo = await  api.Get_instrument_info(instrumentName = "EURUSD");
```

10. Get broker instrument names

```
/**
  * Retrieve broker instrument names
  *
  * @returns {Promise<Object>} - List[boolean, string]
  *       List[0] - bool: status, true or false
  *       List[1] - []: List with broker instrument names
  * @throws {JsTraderApiError} If connection fails.
  */

brokerNames = await api.Get_broker_instrument_names();
```

11. Check instrument in market watch.

```
/**
  * Check if instrument is in market watch.
  *
  * @param {string} [instrumentName] - Name of instrument.
  * @returns {Promise<Object>} - List[boolean, string]
  *       List[0] - bool: status, true or false
  *       List[1] - string: 'Market watch' or 'Not in market watch'
  * @throws {JsTraderApiError} If connection fails.
  */

const inMarketWatch =await  api.Check_market_watch(instrumentName = "EURUSD");
```

12. Check license type.

```
/**
  * Check Mt4/MT5 license type
  *
  * @returns {Promise<Object>} - List[boolean, string]
  *       List[0] - bool: status, true or false
  *       List[1] - string: 'Demo' or 'Licensed'
  * @throws {JstraderApiError} If connection fails.
  */
const checkLicense = await api.Check_license();
```

13. Check trading allowed

```
/**
  * Check if for instrument trading is allowed
  *
  * @param {string} [instrumentName] - Name of instrument.
  * @returns {Promise<Object>} - List[boolean, string]
  *       List[0] - bool: status, true or false
  *       List[1] - 'Allowed or 'Not allowed'
  * @throws {JstraderApiError} If connection fails.
  */
const tradingAllowed = await api.Check_trading_allowed(instrumentName = 'EURUSD');
```

14. Check terminal type.

```
/**
  * Check terminal type MT4 or MT5
  *
  * @returns {Promise<Object>} - List[boolean, string]
  *       List[0] - bool: status, true or false
  *       List[1] - string: MT4 or MT5
  * @throws {JstraderApiError} If connection fails.
  */

const terminalType = await api.Check_terminal_type();
```

15. Check terminal server connection.

```
/**
  * Check if MT terminal is connected to the broker server
  *
  * @returns {Promise<Object>} - List[boolean, string]
  *       List[0] - bool: status, true or false
  *       List[1] - string: 'Connected' or 'Not connected
  * @throws {JstraderApiError} If connection fails.
  */

const serverConnection = await api.Check_terminal_server_connection();
```

16. Get last tick information.

```
/**
   *  Retrieve instrument last tick info.
   *
   *  @param {string} [instrumentName] - Name of instrument.
   *  @returns {Promise<Object>} - List[boolean, dict]
   *       List[0] - bool: status, true or false
   *       List[1] - {instrument, date, ask, bid, last, volume, spread, date_in_ms}
   *  @throws {JstraderApiError} If connection fails or unknown instrument name.
   */
```

lastTick = await api.Get_last_tick_info(instrumentName = "EURUSD")

17. Get actual bar information

```
/**
    * Retrieve instrument last bar/candle info.
    *
    * @param {string} [instrumentName] - Name of instrument.
    * @param {number} [timeFrame=1] - Time frame in MT5 format (integer number.
    * @returns {Promise<Object>} - List[boolean, dict]
    *      List[0] - bool: status, true or false
    *      List[1] - {instrument, date, open, high, low, close, volume}
    * @throws {JstraderApiError} If connection fails, or unknow instrument name.
    */
```

actualBar = await api.Get_actual_candle_info(instrument= "EURUSD",
timeframe= api.get_timeframe_value("H4"))

18. Get specific bars.

```
 /* Retrieve information for specific bar for list of instrument
    *
    * @param {string} [[instrumentName, instrumentName, ..]] - Name of instrument.
    * @param {number} [timeFrame=16408] - Time frame in MT5 format.
    * @returns {Promise<Object>} - List[boolean, List[{}…]
    *      List[0] - bool: status, true or false
    *      List[1] – [{instrument, date, open, high, low, close, volume}, ….]
    * @throws {JstraderApiError} If connection fails or ….
    */
```

specificBars = await api.Get_specific_bars_info(instrumentList = ["EURUSD", "GBPUSD"],
specificBar_index = 5, timeFrame = api.get_timeframe_value("H1"));

19. Get last x ticks from now.

```
/**
    * Retrieve instrument last x tick info.
    *
    * @param {string} [instrumentName] - Name of instrument.
    * @param {number} [nbrOfTicks] - Number of ticks to retrieve.
    * @returns {Promise<Object>} - List[boolean, List[{}]]
    *       List[0] - bool: status, true or false
    *      List[1] - [{dateInMs, ask, bid, last, volume, spread}{}....]
    * @throws {JstraderApiError} If connection fails or ….
    */
```

```
Const ticks = await api.Get_last_x_ticks_from_now(instrumentName = “EURUSD”,
          nbrOfTicks = 100)
```

20. Get last x bars from now.

```
/**
    * Retrieve instrument last x bars/candles info.
    *
    * @param {string} [instrumentName] - Name of instrument.
    * @param {number} [instrumentName] - Time frame.
    * @param {number} [nbrOfBars] - Number of bars to retrieve.
    * @returns {Promise<Object>} - List[boolean, List[{}]]
    *       List[0] - bool: status, true or false
    *       List[1] - [{date, open, high, low, close, volume}{}....]
    * @throws {JsTraderApiError} If connection fails or ....
    */
```

```
Const bars = await api.Get_last_x_bars_from_now(instrumentName = “EURUSD”,
          timeFrame= api.get_timeframe_value(“M1”), nbrOfBars: 1000)
```

21. Open order.

```
/**
  * Open order
  *
  * @param {string} [instrumentName] - name of instrument.
  * @param {string} [orderType] - buy, sell, buy limit, sell limit, buy stop, sell stop.
  * @param {number} [volume] - order volume.
  * @param {number} [openPrice] - for buy and sell, should be 0.0.
  * @param {number} [slippage] - slippage.
  * @param {number} [magicNumber] - magic number.
  * @param {number} [stopLoss] - stop loss. Value 0.0 no stopLoss set.
  * @param {number} [takeProfit] - take profit. Value 0.0 no takeProfit set.
  * @param {string} [comment] - order comment.
  * @param {boolean} [market] - market instrument.
  *
  * @returns {Promise<Object>} - List[boolean, number]
  *      List[0] - bool: status, true or false
  *      List[1] - ticket number, -1 = error
  * @throws {JsTraderApiError} If connection fails or .....
  */
```

## open market

```
Const marketOrder = await api.Open_order(instrumentName = "EURUSD", orderType = "buy",
          volume= 0.01,   openPrice = 0.0, slippage = 10, magicNumber = 2000, stopLoss = 0.0,
          takeProfit  = 0.0, comment = "Test");
```

## open pending order

```
Const pendingOrder = await api.Open_order(instrumentName = "EURUSD", orderType= "buy_stop",
          volume= 0.04,   openPrice = 1.0870, slippage = 10, magicNumber = 2000, stopLoss = 1.0830,
          takeProfit = 1.0950, comment = "Test");
```

22. Set SL and TP for position.

```
/**
   * Set stop loss and take profit for a position.
   *
   * @param {number} [ticket] - ticket of position.
   * @param {number} [stopLoss] - stop loss value, if 0.0 no change.
   * @param {number} [takeProfit] - take profit value, if 0.0 no change.
   *
   * @returns {Promise<Object>} - List[boolean, number]
   *      List[0] - bool: status, true or false
   *      List[1] - number - ticket]
   * @throws {JsTraderApiError} If connection fails or ....
   */
```

```
const modifyPosition = await api.Set_sl_and_tp_for_position(ticket = 53136604, stopLoss = 0.0,
                                        takeProfit = 1.11001);
```

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

```
/**
   * Set stop loss and take profit for a pending order.
   *
   * @param {number} [ticket] - ticket of order.
   * @param {number} [stopLoss] - stop loss value, if 0.0 no change.
   * @param {number} [takeProfit] - take profit value, if 0.0 no change.
   *
   * @returns {Promise<Object>} - List[boolean, number]
   *      List[0] - bool: status, true or false
   *      List[1] - number - ticket]
   * @throws {JsTraderApiError} If connection fails or ....
   */
```

```
const modifyOrder = await api.Set_sl_and_tp_for_order(ticket = 53136804, stopLoss = 0.0,
                              takeProfit = 1.12001);
```

24. Reset stop_loss and take profit for position.

```
/**
   * Reset stop loss and take profit for position.
   * @param {number} [ticket] - ticket of position.
   *
   * @returns {Promise<Object>} - List[boolean, number]
   *      List[0] - bool: status, true or false
   *      List[1] - number - ticket]
   * @throws {JsTraderApiError} If connection fails or ....
   */

const resetSlTp = await api.Reset_sl_and_tp_for_position();
```

25. Reset stop loss and take profit for pending order.

```
/**
   * Reset stop loss and take profit for pending order.
   *
   * @param {number} [ticket] - ticket of order.
   *
   * @returns {Promise<Object>} - List[boolean, number]
   *      List[0] - bool: status, true or false
   *      List[1] - number - ticket]
   * @throws {JsTraderApiError} If connection fails or ....
   */

const resetSlTp = await api.Reset_sl_and_tp_for_pending order();
```

26. Change settings for pending order.

```
/**
   * Change settings of pending order.
   *
   * @param {number} [ticket] - ticket of order.
   * @param {number} [price] - stop loss value, if -1.0 no change.
   * @param {number} [stopLoss] - stop loss value, if -1.0 no change.
   * @param {number} [takeProfit] - take profit value, if -1.0 no change.
   *
   * @returns {Promise<Object>} - List[boolean, number]
   *      List[0] - bool: status, true or false
   *      List[1] - number - ticket]
   * @throws {JsTraderApiError} If connection fails or ....
   */

const changePending = await api.Change_settings_for_pending_order(ticket = 0);
```

27. Delete pending order.

```
/**
 * Delete pending order by ticket.
 *
 * @param {number} [ticket] - ticket of pending order.
 *
 * @returns {Promise<Object>} - List[boolean, number]
 *      List[0] - bool: status, true or false
 *      List[1] - number - ticket]
 * @throws {JsTraderApiError} If connection fails or ....
 */

const deletePending = await api.Delete_pending_order_by_ticket(ticket = 0);
```

28. Close position.

```
/**
 * Close position by ticket.
 *
 * @param {number} [ticket] - ticket of position.
 *
 * @returns {Promise<Object>} - List[boolean, number]
 *      List[0] - bool: status, true or false
 *      List[1] - number - ticket]
 * @throws {JsTraderApiError} If connection fails or ....
 */

const closePosition = await api.Close_position_by_ticket(ticket = 0);
```

29. Close position partial.

```
/**
 * Close open position by opposite position.
 *
 * @param {number} [ticket] - ticket of position to close.
 * @param {number} [oppositeTicket] - opposite ticket for closing position.
 *
 * @returns {Promise<Object>} - List[boolean, number]
 *       List[0] - bool: status, true or false
 *      List[1] - number - ticket]
 * @throws {JsTraderApiError} If connection fails or ....
 */

const closePartial = await api.Close_position_partial_by_ticket(ticket = 0, volumeToClose = 0.01);
```

30. Close position by opposite position(CloseBy).

```
/**
   * Close open position by opposite position.
   *
   * @param {number} [ticket] - ticket of position to close.
   * @param {number} [oppositeTicket] - opposite ticket for closing position.
   *
   * @returns {Promise<Object>} - List[boolean, number]
   *       List[0] - bool: status, true or false
   *       List[1] - number - ticket]
   * @throws {JsTraderApiError} If connection fails or ....
   */
```

const closeBy = await api.Closeby_position_by_ticket(ticket = 0, oppositeTicket = 0);

31. Close positions async.

```
/**
   * Close open positions async, no wait for result.
   *
   * @param {string} [instrumentName] - instrument name
   *                              = '***' , all instruments
   * @param {number} [magicNumber] - additional filter for closing.
   *
   * @returns {Promise<Object>} - List[boolean, null]
   *       List[0] - bool: status, true or false
   *       List[1] - null
   * @throws {JsTraderApiError} If connection fails or ....
   */
```

const closeBy = await api.Close_positions_async(instrumentName = “”, magicNumber = 1000);***

32. Get all pending orders.

```
/**
   * Retrieve all pending orders.

   * @returns {Promise<Object>} - List[boolean, List[{}]]
   *       List[0] - bool: status, true or false
   *       List[1] - [{index, ticket, instrument, order_type, magic_number, volume, open_price,
                 stop_loss, take_profit, comment}{}....]
   * @throws {JsTraderApiError} If connection fails or ....
   */
```

```
const allPendingOrders = api.Get_all_pending_orders();
```

33. Get all deleted pending orders within window.

```
/**
 * Retrieve all deleted orders in window.
 *
 * @param {date}[ dateFrom = new Date('2024-09-18T10:10:10')]  - start date
 * @param {date} [dateTo = new Date.now()] - end date
 * @returns {Promise<Object>} - List[boolean, List[{}]]
 *      List[0] - bool: status, true or false
 *      List[1] - [{index, ticket, instrument, order_type, magic_number, volume, open_price,
 *                    open_time, stop_loss,  delete_price, delete_time, comment }{}....]
 * @throws {JsTraderApiError} If connection fails or ....
 */
```

```
const allDeletedOrders = await api.Get_all_deleted_pending_orders_within_window(
                    (dateFrom = new Date('2024-09-18T10:10:10'), dateTo = new Date());
```

34. Get all deleted pending orders.

```
/**
 * Retrieve all deleted orders.
 *
 * @returns {Promise<Object>} - List[boolean, List[{}]]
 *      List[0] - bool: status, true or false
 *      List[1] - [{index, ticket, instrument, order_type, magic_number, volume, open_price,
 *                    open_time, stop_loss,  delete_price, delete_time, comment }{}....]
 * @throws {JsTraderApiError} If connection fails or ....
 */
```

```
const allDeletedOrders = await api.Get_all_deleted_pending_orders ( );
```

35. Get all open positions.

```
/**
  * Retrieve all open positions.

  * @returns {Promise<Object>} - List[boolean, List[{}]]
  *      List[0] - bool: status, true or false
  *      List[1] - [{index, ticket, instrument, position_type, magic_number, volume, open_price,
  *                  open_time, stop_loss, take_profit, comment, profit, swap, commission}{}....]
  * @throws {JsTraderApiError} If connection fails or ....
  */
```

```
const allOpenPositions = api.Get_all_open_positions();
```

36. Get all closed positions within window.

```
/**
  * Retrieve all closed positions in window.
  *
  * @param {date}[ dateFrom = new Date('2024-09-18T10:10:10')]  - start date
  * @param {date} [dateTo = new Date.now()] - end date
  * @returns {Promise<Object>} - List[boolean, List[{}]]
  *      List[0] - bool: status, true or false
  *      List[1] - [{index, ticket, instrument, position_type, magic_number, volume, open_price,
  *                  open_time, stop_loss, take_profit, close_price, close_time, comment, profit,
  *                  swap, commission}{}....]
  * @throws {JsTraderApiError} If connection fails or ....
  */
```

```
const allClosedPositions = await api.Get_all_closed_positions_within_window(
                    (dateFrom = new Date('2024-09-18T10:10:10'), dateTo = new Date());
```

37. Get all closed positions.

```
/**
  * Retrieve all closed positions.
  *
  * @returns {Promise<Object>} - List[boolean, List[{}]]
  *      List[0] - bool: status, true or false
  *      List[1] - [{index, ticket, instrument, position_type, magic_number, volume, open_price,
  *                  open_time, stop_loss, take_profit, close_price, close_time, comment, profit,
  *                  swap, commission}{}....]
  * @throws {JsTraderApiError} If connection fails or ....
  */
```

```
const allClosedPositions = await api.Get_all_closed_positions ();
```

38. Set global variable.

```
/**
 * Set global variable value.
 *
 * @param {string} [globalName] - name of global variable, if not exists will be created.
 * @param {number} [globalValue] - value of global variable, should be a real.
 *
 * @returns {Promise<Object>} - List[boolean, number]
 *      List[0] - bool: status, true or false
 *      List[1] - string - globalName]
 * @throws {JsTraderApiError} If connection fails or ....
 */
```

```
const setGlobal = await api.Set_global_variable(globalName = 'glbTest', globalValue = 0.0);
```

39. Get global variable.

```
/**
 * Get global variable value.
 *
 * @param {string} [globalName] - name of global variable
 *
 * @returns {Promise<Object>} - List[boolean, number]
 *      List[0] - bool: status, true or false
 *      List[1] - number - global value]
 * @throws {JsTraderApiError} If connection fails or ....
 */
```

```
const getGlobal = await api.Get_global_variable(globalName);
```

40. Switch auto trading on/off.

```
/**
 * Switch auto trading on / off.
 *
 * @param {bool} [onOff] - true = switch on
 *
 * @returns {Promise<Object>} - List[boolean, null]
 *      List[0] - bool: status, true or false
 *      List[1] - null
 * @throws {JsTraderApiError} If connection fails or ....
 */
```

```
const switchOnOf = await api.Switch_auto_trading_on_off(onOff = true);
```