# Elektro backend integration guide

Table of contents

[Elektro backend integration guide 1](#_Toc89870507)

[Introduction 3](#_Toc89870508)

[REST description 3](#_Toc89870509)

[Response format 3](#_Toc89870510)

[Result codes 3](#_Toc89870511)

[HTTP Codes 4](#_Toc89870512)

[Web socket description 4](#_Toc89870513)

[Response format 4](#_Toc89870514)

[Cause codes 4](#_Toc89870515)

[Response types 5](#_Toc89870516)

[Client registration 5](#_Toc89870517)

[Authentication 5](#_Toc89870518)

[REST authentication 5](#_Toc89870519)

[Get session info 6](#_Toc89870520)

[Signature example 6](#_Toc89870521)

[Logout 7](#_Toc89870522)

[Web socket authentication 7](#_Toc89870523)

[Orders submission 7](#_Toc89870524)

[Fields 7](#_Toc89870525)

[Single order funds estimation 8](#_Toc89870526)

[Conditional order funds estimation 8](#_Toc89870527)

[Order payoff estimation 9](#_Toc89870528)

[Single order submission 10](#_Toc89870529)

[Conditional order submission 11](#_Toc89870530)

[Order cancellation 12](#_Toc89870531)

[Order’s status updates 12](#_Toc89870532)

[Client information 12](#_Toc89870533)

[Client fund lock state 12](#_Toc89870534)

[Client open orders 13](#_Toc89870535)

[Client positions 13](#_Toc89870536)

[Order status 14](#_Toc89870537)

[Trade history 14](#_Toc89870538)

[Common information 15](#_Toc89870539)

[System info 15](#_Toc89870540)

[Contract list 15](#_Toc89870541)

[Contract list by ids 16](#_Toc89870542)

[Reference price list 16](#_Toc89870543)

[Spot price list 17](#_Toc89870544)

[Next auction 17](#_Toc89870545)

[Subscribe orderbook 17](#_Toc89870546)

[Unsubscribe orderbook 19](#_Toc89870547)

[Subscribe trade reports 19](#_Toc89870548)

[Unsubscribe trade reports 19](#_Toc89870549)

[Heartbeat request 20](#_Toc89870550)

[Execute backend locally 20](#_Toc89870551)

[Endpoints 20](#_Toc89870552)

[Clients 20](#_Toc89870553)

[Metamask 21](#_Toc89870554)

[Batch time 21](#_Toc89870555)

[Troubleshooting 21](#_Toc89870556)

[Known issues 21](#_Toc89870557)

[Execute test orders 21](#_Toc89870558)

# Introduction

We have REST web service to interact with backend and WebSockets to:

* Receive notifications about order’s statuses
* Subscribe to orderbooks and receive orderbook updates
* Subscribe to trading reports and receive trading reports once batch is performed

# REST description

## Response format

REST response format:

{

"result":"OK",

"details":"",

"payload":{…}

}

* If the result is OK, the payload field contains the appropriate payload, and the details field is empty.
* If the result is not OK, the payload is empty, details may contain additional information.

## Result codes

|  |  |
| --- | --- |
| **Result code** | **Description** |
| OK | Request was successfully processed |
| INVALID\_NONCE | The nonce is invalid |
| INVALID\_SIGNATURE | The signature is invalid |
| UNKNOWN\_ETH\_ADDRESS | Eth address is not registered. Deposit function was not invoked or event has not been caught by the backend yet |
| INVALID\_REQUEST\_DATA | Something is wrong with the request. May be request fields were not filled properly |
| INVALID\_ORDER\_CLASS | Something is wrong with the request. Maybe request fields were not filled correctly, or the request body doesn’t fit the endpoint URL. |
| TRADING\_IS\_NOT\_STARTED\_YET | Need to wait till the system starts or becomes trading hour. |
| INVALID\_ORDER\_SIDE | Side field in the order was not filled properly |
| INVALID\_ORDER\_TYPE | Order type field in the order was not filled properly |
| INVALID\_ORDER\_TIF | Time in force field in the order was not filled properly |
| INVALID\_ACCOUNT | Something is wrong with the account or eth address registered in the backend |
| INVALID\_CONTRACT\_ID | Contract id is invalid, contract does not exist or is not tradeable anymore |
| INVALID\_ORDER\_PRICE\_PRECISION | The price field is filled with incorrect precision. Digits after the point should not exceed backend precision for the currency |
| INVALID\_ORDER\_PRICE | Price field was not set or negative for single order |
| INVALID\_ORDER\_QTY\_PRECISION | The quantity field is filled with incorrect precision. Digits after the point should not exceed backend precision for the currency |
| INVALID\_ORDER\_QTY | Quantity field was not set or negative for single order |
| INVALID\_ORDER\_ID | Order id is invalid or does not exist |
| INVALID\_CN\_VALIDATION\_FAILED | Something went wrong on deeper validation, check details field for more information |
| INVALID\_ORDER\_LEGS | Order legs were not found in conditional order or duplicate in contract ids found. |

## HTTP Codes

|  |  |  |
| --- | --- | --- |
| **Code** | **Name** | **Description** |
| 200 | OK | Response body contains REST response. Request successfully processed |
| 412 | PRECONDITION\_FAILED | Response body contains REST response. Request processed, but error occurred |
| 401 | UNAUTHORIZED | Response body is empty. Authentication required. The session has expired or has not been authenticated |
| 500 | INTERNAL\_SERVER\_ERROR | Unexpected error. Response body is empty |

# Web socket description

## Response format

Web socket response format:

{

"error": true,

"cause": "INVALID\_AUTH\_TOKEN",

"details": "",

"responseType": "VALIDATE\_AUTH\_TOKEN\_RESPONSE",

" payload" : {…}

}

* If the error flag is false, response type contains type of the payload and payload field contains appropriate payload, details and cause are empty.
* If the error flag is true, the payload is empty, details may contain additional information, cause contains error code.

## Cause codes

|  |  |
| --- | --- |
| **Value** | **Description** |
| GENERAL | Something went wrong on the backend |
| INVALID\_REQUEST | Request format is invalid, backend was not able to parse it |
| INVALID\_REQUEST\_ACTION | Request action is invalid |
| INVALID\_CONTRACT\_ID | Contract is not found or not tradeable anymore |
| ALREADY\_SUBSCRIBED | Client already subscribed to requested channel |
| NOT\_SUBSCRIBED | Client was already unsubscribed from requested channel or was not subscribed there at all |
| SERVICE\_NOT\_AVAILABLE\_YET | Need to wait till the system starts or becomes trading hour. |
| INVALID\_AUTH\_TOKEN | Authentication token is invalid |
| RECONNECT\_REQUIRED | Need to disconnect web socket and connect again. |

## Response types

|  |  |
| --- | --- |
| **Value** | **Description** |
| VALIDATE\_AUTH\_TOKEN\_RESPONSE | See Web socket authentication chapter |
| ORDERBOOK | See subscribe orderbook chapter |
| ORDERBOOK\_DELTA | See subscribe orderbook chapter |
| SUBSCRIBE\_ORDER\_BOOK\_RESPONSE | See subscribe orderbook chapter |
| UNSUBSCRIBE\_ORDER\_BOOK\_RESPONSE | See unsubscribe orderbook chapter |
| SUBSCRIBE\_TRADE\_REPORT\_RESPONSE | See Subscribe trade reports chapter |
| UNSUBSCRIBE\_TRADE\_REPORT\_RESPONSE | See Unsubscribe trade reports chapter |
| TRADE\_REPORT | See Subscribe trade reports chapter |
| EXEC\_REPORT | Contains execution report structure, see Trade history chapter |
| HEARTBEAT\_RESPONSE | See Heartbeat request chapter |
| UNKNOWN | Check cause code, there is an error |

# Client registration

All that the client should do to register in Elektro system is deposit some amount to the smart contract.

Elektro backend will catch deposit event through the blockchain logs and register client automatically.

# Authentication

Most actions require authentication. The only things work without authentication are:

* Subscribe to orderbooks and receive orderbook updates
* Subscribe to trading reports and receive trading reports once batch is performed
* Retrieve contracts list

## REST authentication

* Client sends request to the server
* Server sends nonce
* Clients signs nonce
* Client sends signed nonce to the server
* Server validates signature and nonce and authenticates client’s session

Authentication is based on the session. Client’s session is based on HTTP cookie.

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/auth/requestAuth |
| **Request** | {  "ethAddress":"0x7fda7543e01caafd1af39585a156eadb3375d234"  } |
| **Response payload** | {  "nonce":"auth-request-for-0xbd44572e53343a0f003b719cf438c6338bd29d9c-4157310944914892772"  } |
| **Endpoint** | POST /api/v1/auth/validateAuth |
| **Request** | {  "nonce":"auth-request-for-0xbd44572e53343a0f003b719cf438c6338bd29d9c-4157310944914892772",  "signature":" 1B.3B7636212157F57A934164D616EB497ED7C6B799B2934BC74F993A2A0B7220CA.5C8E2CD93DD3B4FFF3CE4E82176967EF0679B66A8C99995E3C5E8B2258A06C97"  } |
| **Response payload** | {  "authToken":"hFKNWhm/KaI9sC6INSPOS7kVRU1MkaEA35TE5eANLyE=",  "clientId" : 1  } |
| **Signature generation details** | |
| **Private key** | 0x0000012001 |
| **Eth address** | 0xbd44572e53343A0f003b719cf438C6338bD29d9C |
| **Message** | {  "types": {  "EIP712Domain": [  {"name": "name", "type": "string"},  {"name": "version", "type": "string"},  {"name": "chainId", "type": "uint256"}  ],  "AuthMessage": [  {"name": "content", "type": "string"}  ]  },  "primaryType": "AuthMessage",  "domain": {  "name": "Elektro",  "version": "0.1",  "chainId": 1337  },  "message": {  "content": "auth-request-for-0xbd44572e53343a0f003b719cf438c6338bd29d9c-4157310944914892772"  }  } |
| **Signature format** | V.R.S |
| **Signature** | 1B.84C55FA150A7AA145B21D2169AE31764B7CBAE768AC13A06613A979F0D7D95E5.3C98405BAF7196F801C11582BBD4D0690D3837F763C6828E22545AD2D05DE0A5 |

## 

## Get session info

Check session state. If the session is not authenticated, function returns UNAUTHORIZED(401).

|  |  |
| --- | --- |
| **Endpoint** | GET /api/v1/auth/getSessionInfo |
| **Request** | Empty request body |
| **Response payload** | {  “clientId” : “1”  } |

### Signature example

|  |  |
| --- | --- |
| **Private key** | 0x75bcd78 |
| **Eth address** | 0x7fda7543e01caafd1af39585a156eadb3375d234 |
| **Nonce** | auth-request-for-0x7fda7543e01caafd1af39585a156eadb3375d234--6257534311083545861 |
| **Bytes to be signed in HEX** | 617574682d726571756573742d666f722d3078376664613735343365303163616166643161663339353835613135366561646233333735643233342d2d36323537353334333131303833353435383631 |
| **Signature V in HEX** | 1C |
| **Signature R in HEX** | DBA4A64802D6EE2D096831CA7AB7FEF785692428B2E116DDFA5C3BBF3CE5DC17 |
| **Signature S in HEX** | 60976FDAC389ED9D9C05022F98789B7935B36E13B5F43346E0484EBCF59440C9 |
| **Signature format** | V.R.S |
| **Signature** | 1C.DBA4A64802D6EE2D096831CA7AB7FEF785692428B2E116DDFA5C3BBF3CE5DC17.60976FDAC389ED9D9C05022F98789B7935B36E13B5F43346E0484EBCF59440C9 |

## Logout

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/auth/logout |
| **Request** | Empty request body |
| **Response payload** | Empty payload |

## Web socket authentication

Use auth token received from REST authentication step

|  |  |
| --- | --- |
| **Request** | {  "action" : "VALIDATE\_AUTH\_TOKEN",  "payload" : {  "authToken" : "hFKNWhm/KaI9sC6INSPOS7kVRU1MkaEA35TE5eANLyE="  }  } |
| **Response** | {  "error" : false,  "responseType" : "VALIDATE\_AUTH\_TOKEN\_RESPONSE",  "payload" : {  "clientId" : 101041  }  } |

# Orders submission

There are 2 order classes:

* Client order – single order ( conditional order with only one leg )
* Conditional order – conditional order with multiple legs

After the order submission, backend will send execution reports through the authenticated client's websocket connections.

## Fields

* clientOrderId should be based on current time in milliseconds left shifted to 10 bits. Last 10 bits might be filled with any value. clientOrderId should be unique, so if multiple orders are generated at the same time some sort of counter. Example:

System.currentTimeMillis() << 10 | cnt++

* Side – BUY or SELL
* Order type – only LIMIT is supported
* Time in force – supported values: DAY, GOOD\_TILL\_CANCEL, IMMEDIATE\_OR\_CANCEL, GOOD\_TILL\_DATE, AT\_AUCTION\_ONLY
* Contract id – id of the contract
* Signature is based on EIP-712 structure

## Single order funds estimation

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/estimateOrderLock |
| **Request** | {  "clientOrderId" : 1604911104000001,  "orderType" : "LIMIT",  "timeInForce" : "GOOD\_TILL\_CANCEL",  "clientEthAddress" : "0x7fda7543e01caafd1af39585a156eadb3375d234",  "totalNetPrice" : 330.0000,  "legs" : [ {  "contractId" : 1,  "side" : "BUY",  "quantity" : 30.00000000  }]  } |
| **Response** | {  "result" : "OK",  "details" : "",  "payload" : {  "currencyPair" : "WBTC/USDC",  "underlierAmount" : 0E-8,  "numeraireAmount" : 330.0000  }  } |

## Conditional order funds estimation

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/estimateOrderLock |
| **Request** | {  "clientOrderId" : 1604911104000001,  "orderType" : "LIMIT",  "timeInForce" : "GOOD\_TILL\_CANCEL",  "clientEthAddress" : "0x7fda7543e01caafd1af39585a156eadb3375d234",  "totalNetPrice" : 110.0000,  "legs" : [ {  "contractId" : 4926,  "side" : "BUY",  "quantity" : 100.00000000,  }, {  "contractId" : 4925,  "side" : "SELL",  "quantity" : 100.00000000,  } ]  } |
| **Response** | {  "result" : "OK",  "details" : "",  "payload" : {  "currencyPair" : "WBTC/USDC",  "underlierAmount" : 0E-8,  "numeraireAmount" : 600110.0000  }  } |

## Order payoff estimation

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/estimateOrderPayoff |
| **Request** | {  "clientOrderId" : 1604911104000001,  "orderType" : "LIMIT",  "timeInForce" : "GOOD\_TILL\_CANCEL",  "clientEthAddress" : "0x7fda7543e01caafd1af39585a156eadb3375d234",  "totalNetPrice" : 110.0000,  "legs" : [ {  "contractId" : 4926,  "side" : "BUY",  "quantity" : 100.00000000,  }, {  "contractId" : 4925,  "side" : "SELL",  "quantity" : 100.00000000,  } ]  } |
| **Response** | {  "result" : "OK",  "details" : "",  "payload" : {  "4000.0000" : "2000.0000",  "4500.0000" : "2100.0000",  }  } |

## Single order submission

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/newOrder |
| **Request** | {  "clientOrderId" : 1676258037557249,  "orderType" : "LIMIT",  "timeInForce" : "GOOD\_TILL\_CANCEL",  "clientEthAddress" : "0xbd44572e53343a0f003b719cf438c6338bd29d9c",  "totalNetPrice" : 330.0000,  "legs" : [ {  "contractId" : 1,  "side" : "BUY",  "quantity" : 30.00000000  }],  "signature" : "1C.273FFD7E9ADA7C3DE33BA4B30958720F565CCBF2EE96353AED8D49687B5D05B8.04E416E3E081EEC7DE4CDC1357740BAA6C48BC5ABC3B761A25BEFA094F890352"  } |
| **Private key** | 0x0000012001 |
| **Eth address** | 0xbd44572e53343A0f003b719cf438C6338bD29d9C |
| **Message** | {  "types": {  "EIP712Domain": [  {"name": "name", "type": "string"},  {"name": "version", "type": "string"},  {"name": "chainId", "type": "uint256"}  ],  "ConditionalOrder": [  {"name": "orderType", "type": "string"},  {"name": "orderId", "type": "uint64"},  {"name": "price", "type": "string"},  {"name": "timeInForce", "type": "string"},  {"name": "address", "type": "string"},  {"name": "leg0", "type": "Leg"}  ],  "Leg": [  {"name": "contractId", "type": "uint32"},  {"name": "side", "type": "string"},  {"name": "quantity", "type": "string"}  ]  },  "primaryType": "ConditionalOrder",  "domain": {  "name": "Elektro",  "version": "0.1",  "chainId": 1  },  "message": {  "orderType": "LIMIT",  "orderId": 1676259483811841,  "price": "330.00000000",  "timeInForce": "GOOD\_TILL\_CANCEL",  "address": "0xbd44572e53343a0f003b719cf438c6338bd29d9c",  "leg0":{"contractId":1,"side":"BUY","quantity":"30.00000000"}  }  } |
| **Signature format** | V.R.S |
| **Signature** | 1C.273FFD7E9ADA7C3DE33BA4B30958720F565CCBF2EE96353AED8D49687B5D05B8.04E416E3E081EEC7DE4CDC1357740BAA6C48BC5ABC3B761A25BEFA094F890352 |

## Conditional order submission

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/newOrder |
| **Request** | {  "clientOrderId" : 1604911104000001,  "orderType" : "LIMIT",  "timeInForce" : "GOOD\_TILL\_CANCEL",  "clientEthAddress" : "0xbd44572e53343a0f003b719cf438c6338bd29d9c",  "totalNetPrice" : 110.0000,  "legs" : [ {  "contractId" : 4926,  "side" : "BUY",  "quantity" : 100.00000000  }, {  "contractId" : 4925,  "side" : "SELL",  "quantity" : 100.00000000  } ]  } |
| **Private key** | 0x0000012001 |
| **Eth address** | 0xbd44572e53343A0f003b719cf438C6338bD29d9C |
| **Note** | Order legs should be sorted by contract id with natural order |
| **Message** | {  "types": {  "EIP712Domain": [  {"name": "name", "type": "string"},  {"name": "version", "type": "string"},  {"name": "chainId", "type": "uint256"}  ],  "ConditionalOrder": [  {"name": "orderType", "type": "string"},  {"name": "orderId", "type": "uint64"},  {"name": "price", "type": "string"},  {"name": "timeInForce", "type": "string"},  {"name": "address", "type": "string"},  {"name": "leg0", "type": "Leg"},{"name": "leg1", "type": "Leg"}  ],  "Leg": [  {"name": "contractId", "type": "uint32"},  {"name": "side", "type": "string"},  {"name": "quantity", "type": "string"}  ]  },  "primaryType": "ConditionalOrder",  "domain": {  "name": "Elektro",  "version": "0.1",  "chainId": 1337  },  "message": {  "orderType": "LIMIT",  "orderId": 1604911104000001,  "price": "110.0000",  "timeInForce": "GOOD\_TILL\_CANCEL",  "address": "0xbd44572e53343a0f003b719cf438c6338bd29d9c",  "leg0":{"contractId":4925,"side":"SELL","quantity":"100.00000000"},"leg1":{"contractId":4926,"side":"BUY","quantity":"100.00000000"}  }  } |
| **Signature format** | V.R.S |
| **Signature** | 1B.C04734C58C7467AF7D04579959D81DDD0B9F7F9E86417AA5E263E1C456D7D123.3BA584CEC49A3328DC11A1FF88AAD912FC3671CC1F8E0CEA9328F332D3315E22 |

## Order cancellation

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/orderCancel |
| **Request** | {  "clientOrderId" : 1604911104000001,  } |

## Order’s status updates

Order’s status update notifications come through the Web socket. Web socket should be authenticated. Statuses:

* REJECTED – Something went wrong, order was rejected. See ordRejReason for details
* PENDING\_NEW – Waiting for validation
* NEW – Validation passed
* CANCELED – Order cancelled
* FILLED – Order matched and was completely filled
* PARTIALLY\_FILLED – Order matched and was partially filled
* PARTIALLY\_FILLED\_REM\_CANCELLED – Order matched and was partially filled, but remaining quantity was cancelled.
* CANCEL\_REJECTED - Order cancellation rejected, order id is invalid or order has been fully filled
* SETTLE\_PENDING – Transaction was submitted to the blockchain. See txHash field to find transaction hash. **Note:** single execution may have multiple SETTLE\_NOTIFICATIONS with different tx hash. It is because we optimize cash movements on the backend and then send them to blockchain. When we have a lot of cash movements in the batch we may split them into multiple blockchain transactions.
* SETTLE\_CONFIRMED – Transaction was successfully performed.
* SETTLE\_FAILED – Transaction was reverted (usually because of block reorg). Transaction submission will be repeated
* SETTLE\_REPLACED – Gas price was too low and transaction pended too long. Backend increased gas price and re-submitted transaction.

# Client information

## Client fund lock state

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/clientFundLockState |
| **Request** | Empty |
| **Response** | {  "result" : "OK",  "details" : "",  "payload" : [ {  "currency" : "USDC",  "orderValue" : 0,  "fundLockValue" : 0,  "settleValue" : 0  }, {  "currency" : "WETH",  "orderValue" : 0,  "fundLockValue" : 0,  "settleValue" : 0  }, {  "currency" : "WBTC",  "orderValue" : 0,  "fundLockValue" : 0,  "settleValue" : 0  } ]  } |

## Client open orders

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/clientOpenOrders |
| **Request** | Empty |
| **Response** | {  "result":"OK",  "details":"",  "payload":[  {  "clientOrderId":1601317888000003,  "batchId":1563787000000,  "orderType":"CONDITIONAL",  "timeInForce":"GOOD\_TILL\_CANCEL",  "totalNetPrice":260000.0000,  "legs":[  {  "contractId":6732,  "side":"SELL",  "price":0.0000,  "quantity":15.00000000,  "filledQty":0E-8,  "cancelledQty":0E-8  },  {  "contractId":2000,  "side":"BUY",  "price":0.0000,  "quantity":20.00000000,  "filledQty":0E-8,  "cancelledQty":0E-8  }  ]  }  ]  } |

## Client positions

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/clientCurrentPositions |
| **Request** | Empty |
| **Response** | {  "result":"OK",  "details":"",  "payload":[  {  "contractId":6732,  "positionsAvgPrice":100.0000,  "positionsQty":-15.00000000  }  ]  } |

## Order status

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/orderStatus |
| **Request** | {  "clientOrderId":1601317888000003  } |
| **Response** | {  "result":"OK",  "details":"",  "payload":{  "orderId":1601317888000003,  "clientId":1003,  "orderStatus":"NEW",  "netPrice":260000.0000,  "details":[  {  "contractId":6732,  "side":"SELL",  "originalQty":15.00000000,  "remainingQty":15.00000000,  "cancelledQty":0E-8,  "price":0,  "totalCost":0.0000  },  {  "contractId":2000,  "side":"BUY",  "originalQty":20.00000000,  "remainingQty":20.00000000,  "cancelledQty":0E-8,  "price":0,  "totalCost":0.0000  }  ],  "conditional":true  }  } |

## Trade history

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/tradeHistory |
| **Request** | {  "expiry":"20211126",  "currencyPair":"WBTC/USDC"  } |
| **Note** | Filter values are optional |
| **Response** | {  "result":"OK",  "details":"",  "payload":[  {  "revDate":1637822220000,  "orderId":1677130009950209,  "clientId":419280940573952,  "orderStatus":"PARTIALLY\_FILLED",  "details":[  {  "contractId":1000012,  "side":"BUY",  "currencyPair":"WBTC/USDC",  "expiry":20211126,  "originalQty":1.00000000,  "remainingQty":0.50000000,  "cancelledQty":0E-8,  "price":1200.0000,  "totalCost":600.0000,  "originalPrice":1200.0000  }  ]  },  …  ]  } |

# Common information

## System info

Does not require authentication

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/systemInfo |
| **Request** | Empty |
| **Response** | {  "result": "OK",  "details": "",  "payload": {  "chainId": 1,  "fundlockAddress": "${env:ETHEREUM\_FUNDLOCK\_ADDRESS}",  "tokenAddress": {  "WBTC": "0x2260fac5e5542a773aa44fbcfedf7c193bc2c599",  "WETH": "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2",  "USDC": "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48"  },  "tokenDecimals": {  "WBTC": 8,  "WETH": 18,  "USDC": 6  },  "currencyPrecision": {  "WBTC": 8,  "WETH": 7,  "USDC": 4  }  }  } |

## Contract list

Does not require authentication

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/contractList |
| **Request** | Empty |
| **Response** | {  "result":"OK",  "details":"",  "payload":[  {  "contractId":1,  "payoff":"Call",  "economics":{  "currencyPair":"WBTC/USDC",  "expiry":20220304,  "strike":10.0000,  "priceCurrency":"USDC",  "qtyCurrency":"WBTC"  }  },  …  ]  } |

## Contract list by ids

Does not require authentication

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/contractListByIds |
| **Request** | {" ids":[1,2,34]} |
| **Response** | {  "result":"OK",  "details":"",  "payload":[  {  "contractId":1,  "payoff":"Call",  "economics":{  "currencyPair":"WBTC/USDC",  "expiry":20220304,  "strike":10.0000,  "priceCurrency":"USDC",  "qtyCurrency":"WBTC"  }  },  …  ]  } |

## Reference price list

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/referencePrices |
| **Request** | {  "expiry" : 20211105,  "currencyPair":"WBTC/USDC" } |
| **Response** | {  "result":"OK",  "details":"",  "payload":[  {  "contractId":1,  "referencePrice":100.0,  "lowRange":90.0,  "highRange":120.0,  "updateAt":"2021-11-03T13:01:30-07:00"  },  ...  ]  } |

## Spot price list

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/spotPrices |
| **Request** | Empty |
| **Response** | {  "result":"OK",  "details":"",  "payload":  {  "WBTC/USDC":60000.0000,  "WETH/USDC":4000.0000,  …  }  } |

## Next auction

|  |  |
| --- | --- |
| **Endpoint** | POST /api/v1/clientapi/nextAuction |
| **Request** | Empty |
| **Response** | {  "result":"OK",  "details":"",  "payload":1638388860000  } |

## Subscribe orderbook

Does not require authentication

|  |  |
| --- | --- |
| **Endpoint** | Web Sockets |
| **Request** | {  "action":"SUBSCRIBE\_ORDER\_BOOK",  "payload":{  "contractId":1  }  } |
| **Response** | {  "error":false,  "responseType":"SUBSCRIBE\_ORDER\_BOOK\_RESPONSE",  "payload":{  "contractId":1  }  } |
| **Note** | * Backend sends whole orderbook and then sends orderbook updates (delta) approximately once per second. * To update orderbook replace size field with the value from delta. * Since the price for conditional order is total, conditional order ASKs and BIDs are summarized in separate field. * Delta num increments monotonically. If the gap is detected, re-subscription is required. |
| **Orderbook** | {  "error" : false,  "responseType" : "ORDERBOOK",  "payload" : {  "contractId" : 2000,  "deltaNum" : 11,  "batchId" : 1563787000000,  "asks":[  {  "price":100,  "size":15  }  ],  "bids" : [  {  "price":95,  "size":30  }  ],  "conditionalAskQty" : 0.0,  "conditionalBidQty" : 20.0  }  } |
| **Orderbook delta** | {  "error" : false,  "responseType" : "ORDERBOOK\_DELTA",  "payload" : {  "contractId" : 2000,  "deltaNum" : 12,  "batchId" : 1563787000000,  "asks":[  {  "price":100,  "size":15  }  ],  "bids" : [  ],  "conditionalAskQty" : 0.0,  "conditionalBidQty" : 10.0  }  } |

## Unsubscribe orderbook

Does not require authentication

|  |  |
| --- | --- |
| **Endpoint** | Web Sockets |
| **Request** | {  "action":"UNSUBSCRIBE\_ORDER\_BOOK",  "payload":{  "contractId":1  }  } |
| **Response** | {  "error":false,  "responseType":"UNSUBSCRIBE\_TRADE\_REPORT\_RESPONSE ",  "payload":{  "contractId":1  }  } |

## Subscribe trade reports

Does not require authentication

|  |  |
| --- | --- |
| **Endpoint** | Web Sockets |
| **Request** | {  "action":"SUBSCRIBE\_TRADE\_REPORTS",  "payload":{  "contractId":1  }  } |
| **Response** | {  "error":false,  "responseType":"SUBSCRIBE\_TRADE\_REPORT\_RESPONSE ",  "payload":{  "contractId":1  }  } |
| **Trade report** | {  "error":false,  "responseType":"TRADE\_REPORT",  "payload":{  "contractId":1,  "batchId":1563787005000,  "tradeDetails":{  "100":5  }  }  } |

## Unsubscribe trade reports

Does not require authentication

|  |  |
| --- | --- |
| **Endpoint** | Web Sockets |
| **Request** | {  "action":"UNSUBSCRIBE\_TRADE\_REPORTS",  "payload":{  "contractId":1  }  } |
| **Response** | {  "error":false,  "responseType":"UNSUBSCRIBE\_TRADE\_REPORT\_RESPONSE ",  "payload":{  "contractId":1  }  } |

## Heartbeat request

Does not require authentication

|  |  |
| --- | --- |
| **Endpoint** | Web Sockets |
| **Request** | {  "action":"HEARTBEAT\_REQUEST",  } |
| **Response** | {  "error":false,  "responseType":"HEARTBEAT\_RESPONSE",  } |

# Execute backend locally

* Install Ganache.
* Add Ganache folder to PATH variable
* Install Gurobi client
* Add Gurobi bin folder to PATH variable
* Create GUROBI\_HOME environment variable
* Create LD\_LIBRARY\_PATH environment variable, it should contain path to the Gurobi bin folder
* It is recommended to install gurobi to the standard place (in Linux /opt/gurobi)
* Install Gurobi license: https://www.gurobi.com/documentation/9.1/remoteservices/licensing.html
* Install Java 11 or higher
* Execute: java -jar ui-test-0.1.011-alpha.jar ganache=true port=8545
* PC should be connected to internet
* Chain id for signing is 1
* Chain id to connect Ganache RPC: 1337

## Endpoints

* REST http://localhost:8078
* WebSocket ws://localhost:8079

## Clients

To simplify development process - we pre-created some accounts:

|  |  |  |
| --- | --- | --- |
| **Private key** | **Funds** | **Deposited funds** |
| 0x0000012001 | 50000USDC, 1WBTC, 10WETH | 50000USDC, 1WBTC, 10WETH |
| 0x0000012002 | 50000USDC, 1WBTC, 10WETH | 50000USDC, 1WBTC, 10WETH |
| 0x0000012003 | 50000USDC, 1WBTC, 10WETH | 50000USDC, 1WBTC, 10WETH |
| 0x0000012004 | 50000USDC, 1WBTC, 10WETH | 50000USDC, 1WBTC, 10WETH |
| 0x0000012005 | 50000USDC, 1WBTC, 10WETH | 0USDC, 0WBTC, 0WETH |
| 0x0000012006 | 50000USDC, 1WBTC, 10WETH | 0USDC, 0WBTC, 0WETH |
| 0x0000012007 | 50000USDC, 1WBTC, 10WETH | 0USDC, 0WBTC, 0WETH |
| 0x0000012008 | 50000USDC, 1WBTC, 10WETH | 0USDC, 0WBTC, 0WETH |

## Metamask

Metamask import key feature may say that the key length is invalid, just add leading zeros, so 0x0000012001becomes 0x0000000000000000000000000000000000000000000000000000000000012001

Configure Metamask network to be Localhost:8545, if it requests chain id set it to 1337.

## Batch time

Batch time is 1 minute.

## Troubleshooting

**If Java application doesn't start, but previous execution worked.**

May be ganache was not closed from previous run, please check running processes.

In Windows process name is **node.exe,** in Linux it should be **ganache** or **ganache-cli**.

Kill running ganache processes and try again.

## Known issues

Metamask doesn't show ERC20 tokens balance even if you import tokens manually.

# Execute test orders

If you need some orders to check trade history or notifications, execute:

java -jar test-client-0.1.025-alpha.jar cli cli\_mode.properties simpleconditionalordertest cancelconditionalordertest