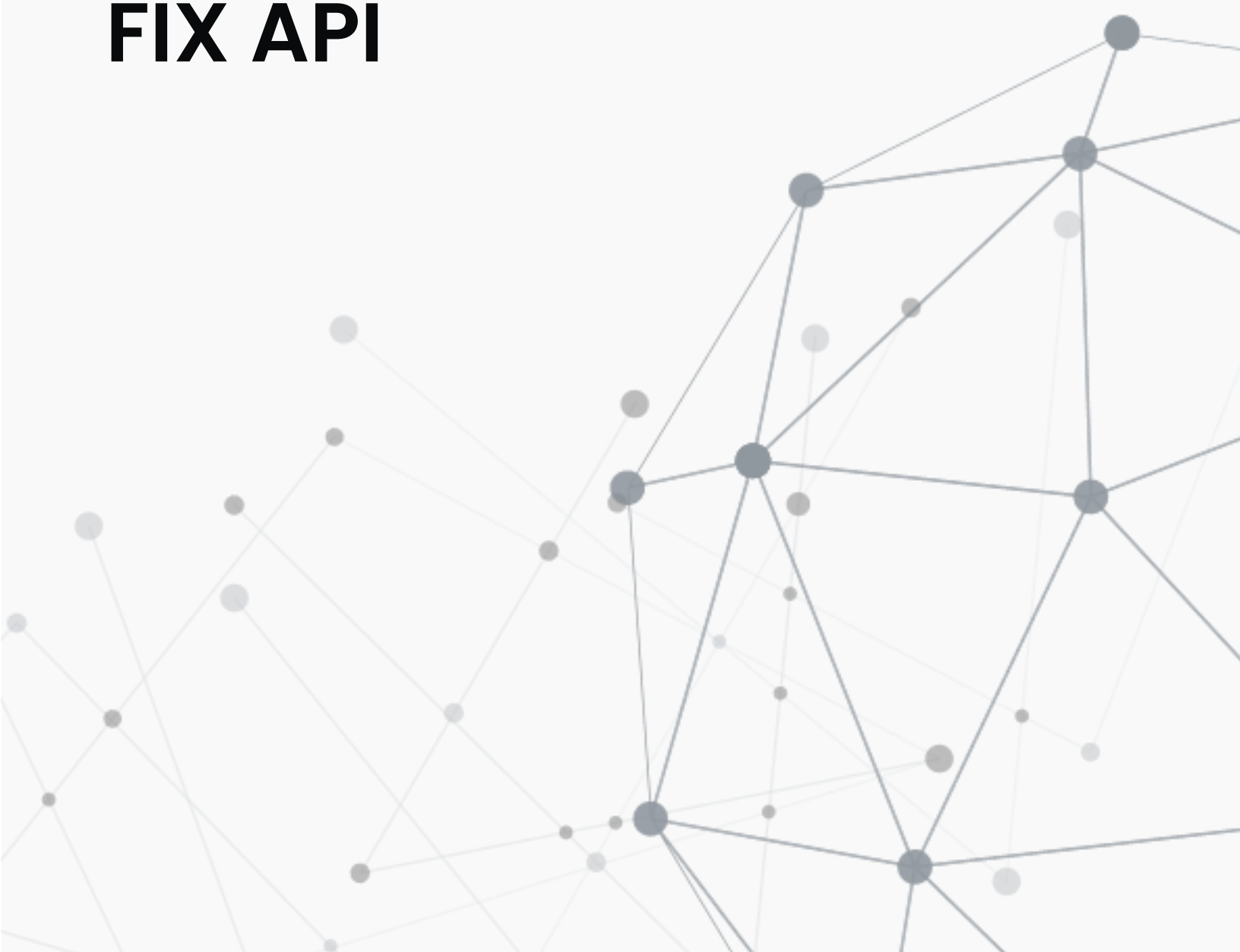
K33 MARKETS

API Documentation
FIX API



K33 Markets API

Overview

The document describes the functionality of the API. This API allows access to K33 Market's trading functions. The examples below are subject to change based on individual deployment.

All calls are made via websocket connection.

Contents

Creating a Web Socket

In order to send and receive JSON messages, you will need to create a Web Socket in your preferred programming language. Below is the sample code for doing so in JavaScript (surrounding code is included as it will be helpful when describing how to log in in the next section).

Web Socket creation in Javascript

```
var websocket = new WebSocket(address);
var serverNonceValue = '';
var cnOnce Value = '';
websocket.onopen = onOpen;
websocket.onclose = onClose;
websocket.onmessage = onMessage;
websocket.onerror = onError;
```

Trading

There are currently five message types you may want to handle in the onMessage event handler in order to process JSON responses from the API. They are 'generalException', 'serverNonce', 'loginResult', 'quoteResponse', and 'tradeResponse'. We will provide an example in the Login section below.

1. Login

In order to log into the API, you can include the sjcl.js source file in the same directory that contains your source code. You will use two functions from that file to create a tokenHash that will be included in your JSON request. You will also make use of the serverNonceValue variable that is shown above as well as the onMessage event handler. Before being able to log in, you will need to use the nonce value that is sent by the API server. You will get a message containing the server nonce value at the start of the connection. (case 'serverNonce').

onMessage event handler in Javascript

```
function onMessage(evt) {
   var jsonMessage = JSON.parse(evt.data);
   switch(jsonMessage.messageType){
      case 'generalException':
         // code
         serverNonceValue = jsonMessage.serverNonce;
         break;
      case 'serverNonce':  //unique value with each connection!
         serverNonceValue = jsonMessage.serverNonce;
         break;
      case 'loginResult':
         //code
```

Now that you have the server nonce value, you can create a client nonce value and tokenHash. (as shown in example code below). For the client nonce value, the example code makes use of Date.getTime(), but there are many ways to create this value. As mentioned above, the serverNonceValue was set at the start of the web socket connection.

JavaScript code for creating a login request

```
const todaysDate = new Date();
cnOnceValue = todaysDate.getTime().toString();
serverNonceValue = serverNonceValue.toString(); //set elsewhere in the code
const password = ' '; //set your password
const bitArray = sjcl.hash.sha256.hash(serverNonceValue + cnOnceValue +
password);
const tokenHash = sjcl.codec.hex.fromBits(bitArray);
const userName = ' '; //set you username
params = { route: 'login', userName: userName, cnOnce : cnOnceValue, token :
tokenHash};
websocket.send(JSON.stringify(params));
```

JSON Request Parameters and Descriptions

Parameter	Type	Description	Example
route	string	Type of message sent	'login'
userName	string	Your username	'assignedUserName'
cnOnce	string	Unique identifier for client	'1605040844949'
token	string	Computed token hash	'4b463ab499a732d0cf7d18f38f8bf037de0052c8e79fcac241b9321629e6924e'

Example JSON request

```
{
  "route": "login",
  "userName": "assignedUserName",
  "cnOnce": 1605060344539,

"token":"d2d798bdc995ac07cf795c4da8e1d83a2449e2910bb606d8c75f654b47840942"
}
```

JSON Response Parameters and Descriptions

Parameter	Type	Description	Example
messageType	string	Type of message	'loginResult'
success	boolean	Whether login request was successful	true
errorDescription	string	Description of error if one occurred	' ' or 'Invalid login'
newNonce	string	New nonce sent from server	' ' or 1605061700976 *used for the next login attempt if unsuccessful
clientIdent	String	Universally unique identifier returned in login response.	1605033314105

Example JSON response for successful login request

{
"messageType": "loginResult",
"success": true,
"errorDescription": "",
"newNonce": ""
"clientIdent": 1605033314105,
}

Example JSON response for unsuccessful login request

{
"messageType": "loginResult",
"success": false,
"errorDescription": "Invalid login",
"newNonce": 1605061700976
}

*Note On any login failure the server will transmit a new nonce value 'newNonce' for the next login attempt

errorDescription return values
'Invalid login'
'No User Profile exists in the user table' - * Admin function
'No account or multiple accounts assigned to this user'
'User ID does not exist'

2. RequestQuote

This is a request for quote (RFQ). After sending the RFQ, the API will return a quote if successful and a price at which a trade can be executed. You can then request execution of a trade using this quote.

JSON Request Parameters and Descriptions

Example JSON request

{
"route": "requestQuote",
"userName": "assignedUserName",
"quantity": 1,
"symbol": "BTC_USD",
"clientIdent": 1605033314105,
"side": "buy"
}

Parameter	Type	Description	Example
route	string	Type of message sent	'requestQuote'
username	string	Your username (case sensitive)	'assignedUserName'
quantity	number	*Quantity in base currency (up to 4 decimal places)	1 or 100
symbol	string	Symbol of instrument for which the quote is being requested	'BTC_USD'
clientIdent	string	Universally unique identifier returned in login response.	1605033314105
side	string	*Indicates side ('buy' or 'sell')	'buy' or 'sell'

JSON Response Parameters and Descriptions

*Note: When you request a quote with a given quantity (e.g. '1') on one side of the market (e.g. 'buy'), the response will return the price of the opposite side of the market ('sell') for that same quantity ('1'), and 0's for both the price and size of the market on the side you passed into the request. A request with 'side': 'buy' and 'quantity': '1' might return something like 'asksz': '1', 'ask': '236.06', 'bidsz': 0, 'bid':

Parameter	Type	Description	Example
messageType	string	Type of message	‘quoteResponse’
success	boolean	Whether request was successful	true
quoteClientID	string	Universally unique identifier, same value that was sent in request	1605033314105
asksz	string	*Size of the ask for requested quote	1
ask	string	Ask price	256.06
bidsz	string	*Size of the bid for requested quote	0
Bid	string	Bid price	0
Symbol	string	Symbol of instrument for which the quote is being requested	‘BTC_USD’
userName	string	Your username	‘assignedUserName’
side	string	Indicates side (‘buy’ or ‘sell’)	‘buy’
tradeKey	string	Unique trade key associated with requested quote	‘61e6f855-f5ba-4526-ad2c-9fd54fb702fd’

Example JSON Response


```
{
  "messageType": "quoteResponse",
  "success": true,
  "quoteResponseID": 1605033314105,
  "asksz": 1.0000000000,
  "ask": 256.06000000,
  "bidsz": 0,
  "bid": 0,
  "symbol": "BTC_USD",
  "userName": "assignedUserName",
  "side": "buy",
  “tradeKey”: "61e6f855-f5ba-4526-ad2c-9fd54fb702fd"
}
```

3. TradeQuote

This is a request to execute a trade. Currently only supporting Fill-Or-Kill (FOK) orders. After sending the RFQ, the API will return a quote if successful and a price at which a trade can be executed. You can then request execution of a trade using this quote. If the trade is executed there will be a response with messageType = 'tradeResponse' and 'success' set to true. If the order is rejected, 'success' will be false.

JSON Request Parameters

Parameter	Type	Description	Example
route	string	Type of message sent	'requestTrade'
clientIdent	string	Universally unique identifier	1604982495730
tradeKey	string	Unique trade key received from quote request	'61e6f855-f5ba-4526-ad2c-9fd54fb702fd'

Example JSON trade request

```
{
  "route" : "requestTrade",
  "clientIdent" : 1604982495730,
  "tradeKey": "61e6f855-f5ba-4526-ad2c-9fd54fb702fd"
}
```

JSON Response Parameters

Parameter	Type	Description	Example
messageType	string	Type of message sent back in response to 'requestTrade'	'tradeResponse'
success	boolean	Whether trade succeeded	true
status	string	Status of the trade	'Filled'
amount	string	Quantity filled by the trade	1
price	number	Price at which the trade executed	265.50
traded	string	Trade id from liquidity provider	1604982586519
symbol	string	Symbol of instrument for which the trade was executed	'BTCUSD.SPOT'
side	string	Indicates side ('buy' or 'sell') of executed trade	'BUY'
clientIdent	string	Universally unique identifier, same value that was sent in request	1604982495730
orderID	number	Internal order ID	1100000000932
errorDescription	string	Error description	'Trade Rejected'

Example JSON response for successful trade request


```
{
  "messageType" : "tradeResponse",
  "success" : true,
  "status" : "Filled",
  "amount" : 1.00,
  "price" : 263.84,
  "tradeID" : 1604982586519,
  "symbol" : "BTCUSD.SPOT",
  "side" :  "buy",
  "clientIdent" : 1604982495730,
  "orderID" : 1100000000932
}
```


Example JSON response for unsuccessful trade request


```
{
  "messageType" : "tradeResponse",
  "success" : false,
  "status" : "Reject",
  "errorDescription" : "TradeRejected"
}
```


4. RequestBalance

Balances you can requested by asses or array of assets. If an empty array is sent, balances for all assets will be returned.

JSON Request Parameters

Parameter	Type	Description	Example
route	string	Type of message sent	'requestBalance'
clientIdent	string	Universally unique identifier	1604982495730
assets	String	Array of assets, if not assets are specified, balances for all available balances will be requested.	['BTC', 'ETH']

Example JSON balance request

{
"route" : "requestBalance",
"clientIdent" : 100200300,
"assets" : [ ]
}

JSON Response Parameters

Parameter	Type	Description	Example
messageType	string	Type of message sent back in response to 'tradeResponse'	'tradeResponse'
success	boolean	Whether trade succeeded	true
symbol	string	Symbol of instrument for which the trade was executed	'BTC'
totalOpenBalance	Number	Amount of asset	100
errorDescription	string	Error description	"Trade Rejected"

Example JSON response for successful trade request

{
messageType: "balanceResponse",
success: true,
balance: [
{
symbol: "BTC",
totalOpenBalance: 100}
},
{
symbol: "ETH",
totalOpenBalance: 200}
},
{
symbol: "ADA",
totalOpenBalance: 300}
}
]
}

Example JSON response for unsuccessful trade request

{
messageType: "orderStatusResponse",
success: false,
errorDescription: "Error Description"
}

5. RequestOrderStatus

Status on an order can be requested by an authenticated user by supplying the ID of the order being requested.

JSON Request Parameters

Parameter	Type	Description	Example
route	string	Type of message sent	'requestOrderStatus'
clientIdent	string	Universally unique identifier	1604982495730
assets	String	Array of assets, if not assets are specified, balances for all available balances will be requested.	['BTC', 'ETH']

Example JSON balance request

{
"route" : "requestOrderStatus",
"clientIdent" : "100200300",
"orderId" : 1100000010123
}

JSON Response Parameters

Parameter	Type	Description	Example
messageType	string	Type of message sent back in response to 'requestOrderStatus'	'requestOrderStatus'
success	boolean	Whether trade succeeded	true
orderID	number	K33 Markets order ID	1100000000932
errorDescription	string	Error description	'Trade Rejected'
status	string	Status of the trade	'Filled'

Example JSON response for successful trade request

```
{
        messageType: "orderStatusResponse",
        success: true,
          order : {
        orderId : 1100000000932,
        status: "New"
   }
}
```

Example JSON response for unsuccessful trade request

```
{
   messageType: "orderStatusResponse",
   success: false,
   errorDescription: "Error Description"
}
```