BitBall DEX API

BitBall DEX's API gives you access to ticker information, orders, trades, deposits, withdrawals, and anything else available through our website. You can also interact directly with the smart contract to get trades, deposits, and withdrawals. For an example of directly working with the contract, take a look at our [contract_observer.py]. BitBall DEX's orderbook is primarily stored offchain, but can be accessed with our public interface or through this API. To learn more about our order system, take a look at our [backend repository].

Websocket Server

BitBall DEX's API is socket.io based. The API utilizes SSL on all connections to keep your data private. You can connect to BitBall DEX's API through this endpoint:

- https://api.bitball-dex.com

Rate Limiting

Please try to limit your calls to the API to a reasonable frequency.

The API currently limits clients to 6 concurrent connections and 12 reconnects per minute per IP address. We reserve the right to adjust these without advanced notice.

The API does not limit the number of `getMarket` and `order` messages the client can send at this time. Protip: subscribe to [this issue](https://github.com/BitBall DEX/proposals/issues/11) to get updates on when that changes.

Clients that violate rate limits repeatedly may be blocked.

Best Practices

Identify your client

Make sure to set a custom User Agent for your BitBall DEX API client whenever possible. The User Agent string should include the name and version of the client, as well as the client's homepage URL and author's contact (email), like so:

A custom User Agent string will allow us to reach out to you in case of any issues and will help us debug issues if you reach out for support.

Backoff on errors

If you receive a one-off disconnect from the server, you may reconnect right away. However, if you receive multiple disconnects or receive errors when attempting to connect (HTTP statuses >=400), you must delay your next connection attempt. We recommend randomized exponential backoff strategy: `retry_delay = min(((2^n)+random_number_milliseconds), 128)`, where `n` is the retry attempt count, starting at 0.

Requests

There are two messages you can send to the BitBall DEX websocket API:

1. getMarket
2. message

1) getMarket (token, user)

getMarket accepts two optional parameters: token & user.

- `token` is the Ethereum address for the token you wish to receive data on.
- `user` is the Ethereum wallet address for the user you wish to receive data on.

When `getMarket` is emitted, a `market` response will be returned.

The **market** response can consist of 6 different parts:

returnTicker

This is standard `returnTicker` information including volume and price data.

Example `returnTicker`:

```
{
   "ETH_0x1183f92":{
      "tokenAddr":"0x1183f92a5624d68e85ffb9170f16bf0443b4c242",
      "quoteVolume":"0",
      "baseVolume":"0",
      "last":"0.0006200000002",
      "bid":"0.00062",
      "ask":"0.001428",
      "updated":"2018-02-21T17:26:11.686344"
   },
   "ETH_0x11f8dd7":{
      "tokenAddr":"0x11f8dd7699147566cf193596083d45c8f592c4ba",
[...]
```

If you would like to get data on each ticker including names, please take a look at our [tokenbase repository](https://github.com/BitBall DEX/tokenbase). If you would just like a simple way to access ticker symbols, take a look at our [configuration JSON](https://BitBall DEX.github.io/config/main.json).

We also supply a REST version of this data, but highly recommend using the websocket server for the most up to date information. The REST version can be found here: https://api.bitball-dex.com/returnTicker

trades | myTrades

If `user` is passed in, `myTrades` will contain all trades available for the passed in user's address. If `user` is not passed in, `myTrades` will be absent.

Example `trades` | `myTrades`:

```
[
  { txHash: '0x75f083bf7a47861dcbb86b30b359de761e57d648c48b5084af7ef3f5db887557',
    date: '2017-10-23T01:16:53.000Z',
    price: '0.219011',
    side: 'sell',
    amount: '70.2',
    amountBase: '15.3745722',
```

```
    buyer: '0xfe988cd30fa97f5422f5a4ae50eafa6271cd2417',
    seller: '0x2056c8184da1fd5a7a1cf43b567c82a999962ef4',
    tokenAddr: '0x8f3470a7388c05ee4e7af3d01d8c722b0ff52374' },
  ...
]
```

orders | myOrders

If `user` is passed in, `myOrders` will contain all orders available for the passed in user's address. If `user` is not passed in, `myOrders` will be absent.

Example `orders` | `myOrders`:

```
{
  sells: [
    { id: '1337b7fe3f96996904d1299fcf030501661158cb964ae6400cbda2ae107978fb_sell',
      user: '0xfb83cB20DFcf7643AbE43Ea23b77F04573eC9616',
      state: '',
      amount: '-2000000000000000000',
      price: '0.9',
      tokenGet: '0x0000000000000000000000000000000000000000',
      amountGet: '1800000000000000000',
      tokenGive: '0x8f3470a7388c05ee4e7af3d01d8c722b0ff52374',
      amountGive: '2000000000000000000',
      expires: '5143967',
      nonce: '893205913',
      v: 28,
      r: '0x4eb35ba40288a169e5f5dfe85a8582db762fde5d57b212afd0be6438ca186f40',
      s: '0x6fc37f071c75c562074b536a354bc79ba3a066f918243fc29813e9a4426b5fa9',
      date: '2017-09-16T11:56:47.006Z',
      updated: '2017-09-16T11:56:47.006Z',
      availableVolume: '1999403702773222212.2222222222222222222',
      ethAvailableVolume: '1.999403702773222',
      availableVolumeBase: '1799463332495900000',
      ethAvailableVolumeBase: '1.7994633324959',
      amountFilled: '0' },
    ...
  ],
  buys: [
    { id: '2c002b763a9aba6d51dbf7274676f7ce957a060bd340b6230aa707fd5ca358a8_buy',
      user: '0xd270fDc1b2a369f890E9858F09E3D0769B63b526',
      state: '',
      amount: '10000000000000000000',
      price: '0.06',
      tokenGet: '0x8f3470a7388c05ee4e7af3d01d8c722b0ff52374',
      amountGet: '10000000000000000000',
      tokenGive: '0x0000000000000000000000000000000000000000',
      amountGive: '600000000000000000',
      expires: '1003884633',
      nonce: '1928222541',
      v: 28,
      r: '0x236d8b8f87163b6dc6712cb90ac85be8eb9fd80d6d671013d8414206d33da1d9',
      s: '0x6813055d05f5300cd45a43ef5e592f78bdc9698e548f3bfb49fc355f327fc92b',
```

```
      date: '2017-09-16T11:56:47.006Z',
      updated: '2017-09-13T14:56:28.838Z',
      availableVolume: '10000000000000000000',
      ethAvailableVolume: '10',
      availableVolumeBase: '600000000000000000',
      ethAvailableVolumeBase: '0.6',
      amountFilled: '0' },
    ...
  ]
}
```

myFunds

If `user` is passed in, `myFunds` will contain deposits and withdrawals for the passed in user's address. If `user` is not passed in, `myFunds` will be absent.

Example `myFunds`:

```
[
  { txHash: '0x295f173773f31c852a9c3eef252f8600620147c6aabb312276f8b0d9800cbc7a',
      date: '2017-10-17T17:36:41.000Z',
      tokenAddr: '0x0000000000000000000000000000000000000000',
      kind: 'Deposit',
      user: '0xcdb1978195f0f6694d0fc4c5770660f12aad65c3',
      amount: '0.001',
      balance: '0.005688612160935313' },
  ...
]
```

2) message (order)

`message` allows you to post an order directly to BitBall DEX and accepts one required parameter.

`order` must be a properly formatted JSON object containing the following properties:

- `amountGive`: the amount you want to give (in wei or the base unit of the token)
- `tokenGive`: the token you want to give (use the zero address, `0x0000000000000000000000000000000000000000` for ETH)
- `amountGet`: the amount you want to get (in wei or the base unit of the token)
- `tokenGet`: the token you want to get (use the zero address, `0x0000000000000000000000000000000000000000` for ETH)
- `contractAddr`: the smart contract address
- `expires`: the block number when the order should expire
- `nonce`: a random number
- `user`: the address of the user placing the order
- `v`, `r`, `s`: the signature of sha256(contractAddr, tokenGet, amountGet, tokenGive, amountGive, expires, nonce) after being signed by the user

On error, emits a `messageResult` event to the originating sid with an array payload, containing:

1. Error code:
    - `400` if the event payload could not be interpreted due to client error (cf. https://httpstatuses.com/400)
    - `422` if the event payload contained semantic errors (cf. https://httpstatuses.com/422)
2. A string error message with a brief description of the problem.
3. An object containing some useful details for debugging.

Example error result:

```
[
    422,
    "Cannot post order because it has already expired",
    { "blockNumber": 5131620, "expires": 5131666, "date": 2017-10-17T17:36:41.000Z }
]
```

On success, emits a `messageResult` event to the originating sid with an array payload, containing:

1. Success code `202`: the order has been accepted.
2. A brief message confirming success

Example success result: `[202, "Good job!"]`

Events

orders

New orders will be emitted as they are placed. The data structure of this `orders` event mirrors that of `market.orders` outlined above. However, some orders will have a `deleted` flag. Orders with the `deleted` flag have been cancelled or traded and are no longer valid.

trades

New trades will be emitted as they occur. The data structure of this `trades` event mirrors that of `market.trades` outlined above.

funds

New deposits and withdrawals will be emitted as they occur. The data structure of this `funds` event mirrors that of `market.myFunds` outlined above.