Namebase Exchange Public API Documentation

The documents below are the official authority on the Namebase Exchange API. Some notes:

- Endpoints *not* enumerated in this document may be discontinued or modified at any time with no notice.
- Error codes will remain consistent within a major API version, but error messages may change at any time with no notice.
- API key authentication may change within a major API version with 1 week of notice/legacy compatibility.
- Rate limiting may change within a major API version with no notice. With all likelihood, we will provide notice, but that is not a guarantee this API provides.

Documents

1. API key authentication instructions
2. Comprehensive list of the REST API endpoints
3. Comprehensive list of error codes and messages
4. Web socket data feed information

Client libraries

Language	Link	Author
Node.js	namebasehq/exchange-api	Namebase
Python	N/A	You? 😀
Go	N/A	You? 😀

Auth for the Namebase API

To obtain an API key, visit the settings page on https://namebase.io/pro and follow the instructions in the user interface.

Be careful with your API keys: they grant unrestricted access to your exchange account.

Authenticating to the API

The Namebase API expects HTTP Basic Auth with a valid pair of Namebase API keys. Use the `ACCESS_KEY` as the username and the `SECRET_KEY` as the password. To authorize your requests, send an HTTP request to one of the API endpoints with the following `Authorization` header:

```
Authorization: Basic /* base64-encoded string of ACCESS_KEY:SECRET_KEY */
```

Here is a code sample of API key authentication using Node.js and the `node-fetch` package:

```
const credentials = Buffer.from(`${ACCESS_KEY}:${SECRET_KEY}`);
const encodedCredentials = credentials.toString('base64');
const authorization = `Basic ${encodedCredentials}`;

fetch(/*ENDPOINT*/, {
  method: 'GET',
  headers: {
    Authorization: authorization,
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
});
```

Note: for Node.js, you may want to use the officially supported Node.js client library linked to in the documentation homepage.

Public Rest API for Namebase

General API Information

- Parameters for GET requests must be sent as a `query string`.
- All other request methods require parameters in the request body as `Content-Type: application/json`
- Parameter order does not matter.
- All timestamps are in milliseconds.
- API errors take the following form:

```
{
  "code": "NAMEBASE_API_ERROR_CODE",
  "message": "Error message appears here."
}
```

Rate limiting

If your account repeatedly gets rate limited, then you are using the API improperly and must back off. Failure to do so will cause your account to lose API access. Namebase retains the right to ban your account with no notice if you are abusing the API.

The rate limits are set generously and are not meant to prohibit any amount of normal usage. In the future, we will standardize our rate limits and provide more explicit feedback about how many remaining requests you can send each minute.

Timing security

- Some endpoints require you to send a `timestamp` of when you created the request.
- You can optionally specify a `receiveWindow` which determines when your timestamped request expires.
- The `receiveWindow` defaults to 5 seconds (so, `5000`)
- If your timestamp is ahead of the server time by too much, you will receive an error complaining about sending a timestamp from the future.
- You can query the exchange info endpoint to synchronize your computer's clock with the exchange's.

Public Namebase API

Terminology

- For a symbol like HNSBTC, HNS is the "base" asset and BTC is the "quote" asset
- All prices are in the units of quote asset per 1 unit of the base asset
- All quantities are in the units of the base asset, and all quote quantities are in the units of the quote asset
- Similarly, volumes are in the units of the base asset, and quote volumes are in the units of the quote asset

ENUM definitions

Order types

- `LMT` (limit)
- `MKT` (market)

Order status

- `NEW` (on the book, no trades yet)
- `PARTIALLY_FILLED` (on the book, some trades have occurred)
- `FILLED` (automatically taken off the book, order fully satisfied)
- `CLOSED` (automatically taken off the book, order partially satisfied)
- `CANCELED` (the trader took the order off the book, order partially satisfied)

When a limit buy order is placed, due to complexities with rounding numbers, it may be the case that your order lacks sufficient quote to be fully satisfied. In this scenario, your order will automatically be removed from the book (the `CLOSED` state) and you will be refunded the dusty quote amount that remained. In the case of BTC, this dust amount is often just a few satoshis. This is rare and requires no action on your part.

When a market order is placed, there may be insufficient liquidity to fully satisfy the order. Alternatively, there may be price slippage and your account may lack sufficient funds to fully satisfy the order. In either case, the order is automtically removed from the book and placed in the `CLOSED` state.

Lastly, if you place two orders on opposite sites and accidentally trade with yourself, then your more recent order, the taker order, will be automatically removed from the book and placed in the

`CLOSED` state. Your older, original order will remain on the book. For your own protection (to save you trading fees), Namebase's matching engine will not match two orders from the same user.

Order side

- `BUY`
- `SELL`

Valid kline intervals:

- `1m` (one minute)
- `5m` (five minutes)
- `15m` (fifteen minutes)
- `1h` (one hour)
- `4h` (four hours)
- `12h` (twelve hours)
- `1d` (one day)
- `1w` (one week)

Miscellaneous endpoints

Exchange information

```
GET /api/v0/info
```

Current exchange trading rules and symbol information. Use to test connectivity to the Rest API.

Parameters: NONE

Response:

```
{
  "timezone": "UTC",
  "serverTime": 1555556529865,
  "symbols": [{
    "symbol": "HNSBTC",
    "status": "TRADING",
    "baseAsset": "HNS",
    "basePrecision": 6,
    "quoteAsset": "BTC",
    "quotePrecision": 8,
    "orderTypes": ["LMT", "MKT"]
  }]
}
```

Market Data endpoints

Order book

```
GET /api/v0/depth
```

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
limit	INT	NO	Default 100; max 1000. Valid limits:[5, 50, 100, 500, 1000]

Response:

```
{
  "lastEventId": 6828,         // The last event id this includes
  "bids": [
    ["0.00003000",  "200.000000"] // [Price level, Quantity]
  ],
  "asks": [
    ["0.00003100", "100.000000"]
  ]
}
```

Trade lookup

```
GET /api/v0/trade
```

Get older trades.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	

Name	Type	Mandatory	Description
symbol	STRING	YES	
tradeId	LONG	NO	Trade id to fetch from. Default gets most recent trades.
limit	INT	NO	Default 100; max 1000.
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
[
  {
    "tradeId": 28457,
    "price": "0.00003000",
    "quantity": "500.000000",
    "quoteQuantity": "0.01500000",
    "createdAt": 1555556529865,
    "isBuyerMaker": true
  }
]
```

Kline data

```
GET /api/v0/ticker/klines
```

Kline (candlestick) bars for a symbol.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
interval	ENUM	YES	
startTime	LONG	NO	Inclusive
endTime	LONG	NO	Inclusive
limit	INT	NO	Default 100; max 1000.

- If startTime and endTime are not sent, the most recent klines are returned.

Response:

```
[
  {

    "openTime": 1557190800000,
    "closeTime": 1557190859999,
    "openPrice": "0.00002247",
    "highPrice": "0.00002256",
    "lowPrice": "0.00002243",
    "closePrice": "0.00002253",
    "volume": "10.001301",
    "quoteVolume": "0.000224824",
    "numberOfTrades": 42
  }
]
```

24hr ticker price change statistics

```
GET /api/v0/ticker/day
```

24 hour rolling window price change statistics.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	

Response:

```
{
  "volumeWeightedAveragePrice": "0.00001959",
  "priceChange": "0.00000019",
  "priceChangePercent": "0.8528",
  "openPrice": "0.00002228",
  "highPrice": "0.00002247",
  "lowPrice": "0.00001414",
  "closePrice": "0.00002247",
  "volume": "11413.935399",
  "quoteVolume": "0.22363732",
  "openTime": 1555467560001,
  "closeTime": 1555553960000
  "closeTime": [illegible],
  "firstTradeId": 19761,
  "lastTradeId": 20926,
  "numberOfTrades": 1166
}
```

Symbol price ticker

```
GET /api/v0/ticker/price
```

Latest price for a symbol or symbols.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	

Response:

```
{
  "price": "0.00002300"
}
```

Symbol order book ticker

```
GET /api/v0/ticker/book
```

Best price/quantity on the order book for a symbol or symbols.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	

Response:

```
{
  "bidPrice": "0.00002000",
  "bidQuantity": "100.000000",
  "askPrice": "0.00002300",
  "askQuantity": "9000.100000"
```

```
    "askQuantity": "0000.100000"
}
```

Account endpoints

New order

```
POST /api/v0/order
```

Send in a new order.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
side	ENUM	YES	Buy or sell etc
type	ENUM	YES	Limit or market etc
quantity	DECIMAL	YES	
price	DECIMAL	NO	Required for limit orders
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
{
  "orderId": 174,
  "createdAt": 1555556529865,
  "price": "0.00000000",
  "originalQuantity": "1000.00000000",
  "executedQuantity": "1000.00000000",
  "status": "FILLED",
  "type": "MKT",
  "side": "SELL",
  "fills": [
    {
      "price": "0.00003000",
      "quantity": "500.000000",
      "quoteQuantity": "0.01500000",
      "commission": "0.00000750",
      "commissionAsset": "BTC"
```

```
        "commissionAsset": "BTC"
      }
      {
        "price": "0.00002000",
        "quantity": "500.000000",
        "quoteQuantity": "0.01000000",
        "commission": "0.00000500",
        "commissionAsset": "BTC"
      }
    ]
  }
```

Query order

```
GET /api/v0/order
```

Check an order's status.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
orderId	LONG	YES	
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
{
  "orderId": 1,
  "price": "0.1",
  "originalQuantity": "1.0",
  "executedQuantity": "0.0",
  "status": "NEW",
  "type": "LMT",
  "side": "BUY",
  "createdAt": 1555556529865,
  "updatedAt": 1555556529865
}
```

Cancel order

```
DELETE /api/v0/order
```

Cancel an active order.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
orderId	LONG	YES	
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
{
  "orderId": 28,
  "price": "1.00000000",
  "originalQuantity": "910.00000000",
  "executedQuantity": "19.00000000",

  "status": "CANCELED",
  "type": "LMT",
  "side": "SELL",
  "createdAt": 1555556529865,
}
```

Current open orders

```
GET /api/v0/order/open
```

Get the most recent open orders on a symbol (limited to 500).

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
receiveWindow	LONG	NO	

Name	Type	Mandatory	Description
timestamp	LONG	YES	

Response:

```
[
  {
    "orderId": 1,
    "price": "0.1",
    "originalQuantity": "1.0",
    "executedQuantity": "0.0",
    "status": "NEW",
    "type": "LMT",
    "side": "BUY",
    "createdAt": 1555556529865,
    "updatedAt": 1555556529865
  }
]
```

All orders

```
GET /api/v0/order/all
```

Get all account orders; active, canceled, or filled.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
orderId	LONG	NO	Greater than or equal to 0.
limit	INT	NO	Default 100; max 1000.
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Notes:

- If `orderId` is set, it will get orders >= that `orderId`. Otherwise you will receive the most recent orders.

Response:

```
[
  {
    "orderId": 1,
    "price": "0.1",
    "originalQuantity": "1.0",
    "executedQuantity": "0.0",
    "status": "NEW",
    "type": "LMT",
    "side": "BUY",
    "createdAt": 1555556529865,
    "updatedAt": 1555556529865
  }
]
```

Account information

```
GET /api/v0/account
```

Get basic account information.

Parameters:

Name	Type	Mandatory	Description
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
{
  "makerFee": 15, // in basis points, 0.15%
  "takerFee": 15, // in basis points, 0.15%
  "canTrade": true,
  "balances": [
    {
      "asset": "HNS",
      "unlocked": "779.900092",
      "lockedInOrders": "100.000000",
      "canDeposit": true,
      "canWithdraw": true
    },
    {
      "asset": "BTC",
      "unlocked": "5.10000012",
      "lockedInOrders": "1.000000".
```

```
        "canDeposit": true,
        "canWithdraw": true
      }
    ]
  }
```

Account withdrawal limits

```
GET /api/v0/account/limits
```

Retrieve your account's withdrawal limits for all assets. Withdrawal limits are applied on a 24-hour rolling basis, and the start and end time of this period are specified by `startTime` and `endTime` .

The `totalWithdrawn` key specifies how much of an asset you have withdrawn in the last 24 hours, and the `withdrawalLimit` key specifies the maximum amount you are permitted to withdraw in the specified period.

Parameters:

Name	Type	Mandatory	Description
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
  {
    "startTime": 1555467560001,
    "endTime": 1555553960000,
    "withdrawalLimits": [
      {
        "asset": "HNS",
        "totalWithdrawn": "500.000000",
        "withdrawalLimit": "10000.000000",
      },
      {
        "asset": "BTC",
        "totalWithdrawn": "0.50000000",
        "withdrawalLimit": "5.00000000",
      }
    ]
```

```
}
```

Account trade list

```
GET /api/v0/trade/account
```

Get trades for a specific account and symbol.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
tradeId	LONG	NO	TradeId to fetch from. Default gets most recent trades.
limit	INT	NO	Default 100; max 1000.
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Notes:

- If `tradeId` is set, it will get trades >= that `tradeId`. Otherwise you will get your most recent trades.

Response:

```
[
  {
    "tradeId": 10921,
    "orderId": 61313,
    "price": "8.00000000",
    "quantity": "200.000000",
    "quoteQuantity": "1600.00000000",
    "commission": "4.500000",
    "commissionAsset": "HNS",
    "createdAt": 1555556529865,
    "isBuyer": true,
    "isMaker": false,
  }
]
```

Order trade list

```
GET /api/v0/trade/order
```

Get trades for a specific order and symbol.

Parameters:

Name	Type	Mandatory	Description
symbol	STRING	YES	
orderId	LONG	YES	OrderId to get the trades for.
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
[
  {
    "tradeId": 10921,
    "orderId": 61313,
    "price": "8.00000000",

    "quantity": "200.000000",
    "quoteQuantity": "1600.00000000",
    "commission": "4.500000",
    "commissionAsset": "HNS",
    "createdAt": 1555556529865,
    "isBuyer": true,
    "isMaker": false,
  }
]
```

Deposit Address

```
POST /api/v0/deposit/address
```

Generate a deposit address.

Parameters:

Name	Type	Mandatory	Description

Name	Type	Mandatory	Description
asset	STRING	YES	
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
{
    "address": "ts1qjg8chhk2t4zff4ltdaug3g9f7sxgne98jyv6ar",
    "success": true,
    "asset": "HNS"
}
```

Withdraw

```
POST /api/v0/withdraw
```

Submit a withdraw request.

Parameters:

Name	Type	Mandatory	Description
asset	STRING	YES	
address	STRING	YES	
amount	DECIMAL	YES	
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
{
  "message": "success",
  "success": true,
  "id": "df7282ad-df8c-44f7-b747-5b09079ee852"
}
```

Deposit History

```
GET /api/v0/deposit/history
```

Fetch deposit history.

Parameters:

Name	Type	Mandatory	Description
asset	STRING	YES	
startTime	LONG	NO	
endTime	LONG	NO	
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
[
  {
    "asset": "HNS",
    "amount": "31.853300",
    "address": "ts1qtq6ymgcep8mz2ag32ftrktwws0hr4uygprjurf",
    "txHash": "e7714680a4d93e3b29348eab38c22bb99949ed4d8aea7006091ff5f9712d1ec6",
    "createdAt": 1555556529865,
  },
  {
    "asset": "HNS",
    "amount": "210.000333",
    "address": "n1M5Rw3r7WkujB2dG1L84M3a4pzr2NKvfp",
    "txHash": "1d0827c642bd67781f80fe15c0fbb349aa4e35117adba06a52add4b207d334dd",
    "createdAt": 1555556529865,
  }
],
```

Withdraw History

```
GET /api/v0/withdraw/history
```

Fetch withdraw history.

Parameters:

Name	Type	Mandatory	Description
asset	STRING	YES	
startTime	LONG	NO	
endTime	LONG	NO	
receiveWindow	LONG	NO	
timestamp	LONG	YES	

Response:

```
[
  {
    "id": "3333edc6-e5c6-4d23-bf84-7b1072a90e37",
    "asset": "HNS",
    "amount": "1.000000",
    "minerFee": "0.100000",
    "address": "ts1qtq6ymgcep8mz2ag32ftrktwws0hr4uygprjurf",
    "txHash": "e7714680a4d93e3b29348eab38c22bb99949ed4d8aea7006091ff5f9712d1ec6",
    "createdAt": 1555556529865,
  },
  {
    "id": "180ceb4d-d303-4fed-9af6-213b5137255a",
    "asset": "HNS",
    "amount": "1200.000000",
    "minerFee": "0.200000",
    "address": "ts1qygv5nh38e9sl8npm4pcx8mqqqfp9sjaq4jrsn5",
    "txHash": "c5c398802554b861bef2ec7c4805846ff400a90f71059619974685848bbc4fd3",
    "createdAt": 1555556529865,
  }
]
```

Error codes for Namebase

Errors consist of two parts: an error code and a message. Codes are universal, but messages can vary. Here is the error JSON payload:

```
{
  "code": "NAMEBASE_API_ERROR_CODE",
  "message": "Error message appears here."
}
```

General server or network issues; unclear fix

SERVER_UNKNOWN

- An unknown server error occured.

SERVER_FUTURE_TIMESTAMP

- The provided timestamp is too far into the future.

SERVER_LATE_TIMESTAMP

- This request was received after the specified receive window.

Request issues; fixed by changing a parameter

REQUEST_PARSE_ERROR

- An unknown error occured while parsing your request parameters.

REQUEST_MISSING_PARAMETER

- Parameter ${key} is required but did not appear in the request.

REQUEST_BAD_PARAMETER

- Parameter ${key} does not adhere to the spec.

REQUEST_EXTRA_PARAMETER

- Parameter ${key} should not have been included.

REQUEST_UNSUPPORTED_SYMBOL

- The Namebase exchange does not yet support symbol "${symbol}".

REQUEST_UNSUPPORTED_ASSET

- The Namebase exchange does not yet support asset "${asset}".

REQUEST_UNSUPPORTED_LIMIT

- The only supported limits are ${acceptedLimitsString}.

REQUEST_MINIMUM_WITHDRAWAL

- The minimum withdrawal for ${asset} is ${amount}.

Stateful errors; hard or impossible for user to fix

NOT_ALLOWED_TO_TRADE

- Your account is not currently allowed to trade. Complete KYC verification or contact support.

NOT_ALLOWED_TO_DEPOSIT

- Unfortunately, based on your verification status, you cannot deposit ${asset}.

NOT_ALLOWED_TO_WITHDRAW

- Unfortunately, based on your verification status, you cannot withdraw ${asset}.

WITHDRAWAL_LIMIT_REACHED

- You've reached your withdrawal limit for ${asset}. Try withdrawing less funds or try again later.

INSUFFICIENT_BALANCE

- You do not have sufficient balance to perform this action.

CANCEL_INVALID

You can only cancel active orders.

CANCEL_UNAUTHORIZED

You do not have permission to cancel this order.

CANCEL_UNKNOWN

Could not cancel this order due to an unknown error.

GET_ORDER_FAILED

Could not retrieve this order.

GET_TRADES_UNAUTHORIZED

You do not have permission to retrieve this order's trades.

Web Socket Data Feeds for Namebase

The host for the streams is wss://namebase.io:443/

The `/ticker` feeds send data every second.

The `/stream` feeds send data on each event as soon as it happens, e.g. on each trade.

Trade stream

The Trades feed sends trade information with the buyer and seller ids scrubbed out.

Feed name: '/ws/v0/ticker/trades' (in flux)

Payload:

```
{
  "eventType": "trade",       // Event type
  "eventTime": 1555553960000, // Event time
  "symbol": "HNSBTC",         // Symbol
  "tradeId": 12345,           // Trade id
  "price": "0.001",           // Price
  "quantity": "100",          // Quantity
  "quoteQuantity": "100",     // Quote quantity
  "createdAt": 1555553960000, // The time the trade occurred

  "isBuyerMaker": true,       // True iff the buy order is the maker
}
```

Klines ticker

Connect to the klines feed to receive klines (i.e. candlesticks) every second.

Valid kline intervals:

- `1m` (one minute)
- `5m` (five minutes)
- `15m` (fifteen minutes)
- `1h` (one hour)
- `4h` (four hours)
- `12h` (twelve hours)

- `1d` (one day)
- `1w` (one week)

Feed name: '/ws/v0/ticker/kline_<interval, e.g. "1m">'

Payload:

```
{
  "eventType": "kline",          // Event type
  "eventTime": 1555553960000,     // Event time
  "symbol": "HNSBTC",            // Symbol
  "kline": {
    "interval": "1m",            // Interval duration
    "isClosed": false,           // Is this kline still being updated?
    "openTime": 1557190800000,    // Kline start time
    "closeTime": 1557190859999,   // Kline close time
    "openPrice": "0.00002247",    // Open price
    "highPrice": "0.00002256",    // High price
    "lowPrice": "0.00002243",     // Low price
    "closePrice": "0.00002253",   // Close price
    "volume": "10.001301",        // Total base asset traded in interval
    "quoteVolume": "0.000224824", // Total quote asset traded in interval
    "numberOfTrades": 42,         // Number of trades
    "firstTradeId": 13001,        // First trade id
    "lastTradeId": 13042,         // Last trade id
  }
}
```

Price ticker

This feed provides statistics for the last 24 hours of trading activity for a HNSBTC, every second. Importantly, these statistics are taken over a *rolling* window and not over a calendar day. Thus, a request sent on `2019/05/05 16:01:33` will return statistics for trades that occurred between `2019/05/04 16:01:34` and `2019/05/05 16:01:33`. The seconds are inclusive on both ends.

Feed name: '/ws/v0/ticker/day'

Payload:

```
{
  "eventType": "24hr",                         // Event type
  "eventTime": 1555553960000,                  // Event time
  "symbol": "HNSBTC",                          // Symbol
  "volumeWeightedAveragePrice": "0.00001959", // Volume weighted average price
  "priceChange": "0.00000019",                // Price change
  "priceChangePercent": "0.8528",             // Price change percent
```

```
    "openPrice": "0.00002228",                    // Open price
    "highPrice": "0.00002247",                    // High price
    "lowPrice": "0.00001414",                     // Low price
    "closePrice": "0.00002247",                   // Close price
    "volume": "11413.935399",                     // Total base asset traded in interva
    "quoteVolume": "0.22363732",                  // Total quote asset traded in interv
    "openTime": 1555467560001,                    // Time these statistics began (inclu
    "closeTime": 1555553960000,                   // Time these statistics end (inclusi
    "firstTradeId": 19761,                        // First trade id
    "lastTradeId": 20926,                         // Last trade Id
    "numberOfTrades": 1166                        // Total number of trades
    "bestBidPrice": "0.00002079",                 // (TODO) Best bid price
    "bestBidQty": "10.130000",                    // (TODO) Best bid quantity
    "bestAskPrice": "0.00002091",                 // (TODO) Best ask price
    "bestAskQty": "32.901102",                    // (TODO) Best ask quantity
  }
```

Depth ticker

This feed allows you to mirror the Namebase order book on your computer. It sends price and quantity updates every second. Note: this API provides no guarantees on the order the price levels appear in. Do not assume that bids and asks are sorted in decreasing order by price.

Feed name: '/ws/v0/ticker/depth'

Payload:

```
  {
    "eventType": "depthUpdate",     // Event type
    "eventTime": 1555553960000,     // Event time
    "symbol": "HNSBTC",             // Symbol
    "firstEventId": 256,            // First event id
    "lastEventId": 258,             // Last event id
    "asks": [
      ["0.00002091", "32.901102"] // [Price level to change, New quantity]
    ],
    "bids": [
      ["0.00002079", "10.130000"] // [Price level to change, New quantity]
    ]
  }
```

Buffering depth updates

Since this feed only provides updates (it's a differential feed), you need to obtain the current orderbook state by querying the `GET /api/v0/depth` REST endpoint. However, it's important that you don't do this *first*.

To properly maintain a local copy of the Namebase orderbook, begin by connecting to the depth feed and storing all of the depth update events in a buffer. Only once you've successfully connected should you query the REST endpoint to get the current orderbook.

With this in hand, you can review your buffer and apply all of the updates that happend *after* the REST response's `lastEventId` . This procedure ensures that you do not miss any events.