# BTC Markets API (3.0.0)

API Support: support@btcmarkets.net | URL: http://btcmarkets.net

# Introduction

Welcome to BTC Markets API v3.

## Overview

Over the past few years, we have received many feedbacks from our customers about their experience using existing APIs in v1 and v2. With API v3, we have tried to address shortcomings in the previous generation and also focus on improving experience for market markers, traders, institutions, and anyone building clients on top of the API.

This new API while covers all functionality currently provided via API v1 and v2 it also offers significant improvements compared to previous API generations including:

| Improvement | Description |
|---|---|
| Compatibility with common REST API guidelines | Easier to develop apps using standard libraries in any programming languages |
| Batch processing | Allowing traders to combine multiple order placements and cancellations in a single Http request |
| Support for ClientOrderId | Allowing traders to use their internal order id hence better tracking of orders |
| Simple and flexible APIs | e.g., no need to make multiple API calls per market. same /orders API returns open or historical orders |

| Improvement | Description |
|---|---|
| Using Http status code for error handling | |
| Easier pagination | |
| New features | e.g., reports API, full orderbook data, multiple market tickers etc |
| Standard data formats | e.g., amount as string, date/time as ISO 8601, etc |
| Better documentation | |

# General notes

### API endpoint

Base URL for all public and private APIs: https://api.btcmarkets.net

Path for all APIs start with `/v3`. For instance the `time` API would be accessed via `https://api.btcmarkets.net/v3/time`

### Request/response format

All requests and responses use `application/json` content type for all APIs.

### Date/time format

All APIs return date/time format in ISO 8601 with microseconds. sample time returned: `2019-08-20T06:22:11.123456Z`

### Number format

Unless specified, all amounts are formatted as `string`. For instance when placing a new order, pass `"1.45"` as the amount to buy 1.45 of `LTC`

At this stage, we support up to 8 digits of decimal points when processing orders, trades, and transfers.

Every market has a configuration for the number of decimal points used for pricing. For instance, for the `BAT-AUD` market you can specify up to 4 decimal points for `price` and for `amount` you can specify up to 8 decimal points. Please refer to the `/v3/markets` API for more information.

### Sort order

By default, all lists are sorted descending, meaning the most recent records appear first in the list.

### Null values

Data objects returned by the Http response will not include attributes with null values. For instance, if you are not using clientOrderId for placing orders, then the data object returned by Http response won't include it.

APIs defined in this document have sample response format and in some cases also an option to view the list of all possible attributes (from the right panel)

### Record Ids

Records generated by the exchange will have ids generated and represented in `string` format.

At this stage, some of the record ids generated by the exchange might have numbers only; however, we can't guarantee that they continue to use numbers only as we continue to change our underlying systems.

Therefore we recommend all client apps to use string format to represent record ids to avoid bugs at a later stage.

### Rate Limits

All APIs are rate limited, and we currently measure rates over 10 seconds intervals. The default rate limit (unless specified) for public and private APIs is up to `50 calls per 10 seconds`. There are exceptions to this rule as per below:

| Name | API | Limit |
|---|---|---|
| order placement API | /v3/orders POST | 30 calls per 10 seconds |
| batch order API | /v3/batchorders POST | 5 calls per 10 seconds |
| withdraw request API | /v3/withdrawals POST | 10 calls per 10 seconds |
| creating new report API | /v3/reports POST | 1 call per 10 seconds |

### HTTP status

All APIs return commonly used HTTP statuses (e.g. 400, 401, 403, etc.) for errors and HTTP status 200 for a successful response.

## API client libraries

### Sample code

If you are starting to develop your client app, then the following samples can be useful as they demonstrate authentication for varies types of requests.

- Node.js: https://github.com/BTCMarkets/api-v3-client-node
- .NET(C#): https://github.com/BTCMarkets/api-v3-client-dotnet
- Python: https://github.com/BTCMarkets/api-v3-client-python
- Java: https://github.com/BTCMarkets/api-v3-client-java
- Go: https://github.com/BTCMarkets/api-v3-client-go
- Swift: https://github.com/BTCMarkets/api-v3-client-swift

### Client SDK

Client SDKs encapsulate the underlying API calls allowing you to focus on the business logic of your application.

coming soon...

### Community SDKs and Libraries

Thanks to our active community of trading engineers.

If you develop client libraries that you believe others in the community might find useful, please let us know, and we will list them as part of this document.

- Ruby: https://github.com/2pd/btcmarkets-ruby

### Help from the community

if you have a technical question, someone else might have asked and solved the same problem.

Please check out past issues here and feel free to raise new questions or issues here:

https://github.com/BTCMarkets/API/issues

### Previous versions of API

You can find information about previous versions of this API here:

https://github.com/BTCMarkets/API#api-v1v2

# Error handling

By default, all successful requests result in HTTP `200` status and with appropriate entity is being returned.

If any issues occur in processing the request, then a response code will be returned and a body that contains a JSON message with more information about the underlying problem.

For instance, placing an order with an invalid amount (e.g., too low) will result in Http 400 error and a response body of the following:

`{"code":"InvalidOrderAmount", "message":"invalid order amount"}`

We would encourage client app developers to use appropriate Http frameworks/libraries that allow them to capture the body of the response for handling of errors.

## HTTP status codes

The following HTTP status codes are supported, and we make our best effort to send one of the following responses in any situation.

| **Http response** | **Description** |
|---|---|
| 200 - OK | The HTTP request was successful. |
| 400 - Bad Request | The request failed due to a syntax error, or is missing a required parameter |
| 401 - Unauthorized | Invalid API key or message signature generation issue |
| 403 - Forbidden | You do not have access to the requested resource |
| 404 - Not Found | The requested resource not found |
| 429 - Too Many Requests | You have exceeded throttle |
| 500 - Internal Server Error | We failed to process your request due to a server problem |
| 502 - Bad Gateway | Exchange is down or is under maintenance |

## API Error codes

While handling HTTP status codes covers requirements for most cases and client apps. However, there are situations where having a machine-readable and specific error code would help handling specific cases more appropriately.

For instance, when an account runs out of funds then calling API to place an order would return HTTP 400 and an error message as `Insufficient fund`. A client app may categorize all HTTP 400s as an indication of an issue and halt running or log the issue for a person to investigate and fix it. However, some apps may have application logic specifically to deal with runnning out of funds issue and this case the app would parse the text returned by http response `Insufficient fund` and determines that it needs to make a deposit.

Using the text message to make that decision might be difficult given that the text requires parsing, and in case if the text message is updated in the future (e.g., updated to `Insufficnet fund in your ETH wallet`) it may break the client application logic.

As a result, an additional error code is provided (along with the `message`) inside the body of the HTTP response that can be used to handle specific situations programmatically.

Below is a sample of such error codes:

`{"code":"IncorrectDecimalPoints","message":"incorrect number of decimal points for price"}`

Please refer to this section for full list of errors: **Full list of error codes**

We ensure that those error codes remain the same for the lifetime of the API. However, the text message that comes with errors may change.

Also, some of the generic codes (e.g., BadRequest) may be changed to be specific codes (e.g., InvalidAmount) in the future (but not the other way around).

# Authentication

## Generate API key

Before you begin, please make sure to generate API keys on the website. API keys come with permission for each area (e.g., trading, account and fundtransfer)

## Authentication parameters

In order to authenticate your request, the following parameters are required:

- API key
- API private key
- Http method: `POST`, `GET` or `DELETE`
- Current timestamp in millisecond (you check server time by calling API `/v3/time` )
- Path of the request (with no query string)

## Authentication process

Use the following steps:

- Build a string message: `method + path + timestamp + data`
- Sign the message using `Hmac512` algorithm
- Add the following HTTP headers: `BM-AUTH-APIKEY`, `BM-AUTH-TIMESTAMP` and `BM-AUTH-SIGNATURE`

## Build message signature

The message to sign is a string concatenation of the HTTP method, API path, timestamp, and post data.

The following code sample builds the message for authenticating for `/v3/reports` API using HTTP `POST`.

```
var timestamp = Date.now();
var path = '/v3/reports';
var data = JSON.stringify({type:"TransactionReport",format:"json"});
var message =  "POST" + path + timestamp + data;
```

- Replace `POST` with `GET` or `DELETE` for `GET/DELETE` APIs.
- For `GET` and `DELETE` requests the data portion of the message has no value; hence, it is not added to the message for signing.

## Sign the message

The following code sample signs the message for authentication.

```
var buffer = Buffer.from(apiPrivateKey, 'base64');
var hmac = crypto.createHmac('sha512', buffer);
var signature = hmac.update(message).digest('base64');
```

## Set http headers

The following code sample sets HTTP headers for authentication.

```
var headers = {
  "Accept": "application/json",
  'Content-Length': Buffer.byteLength(data),
  "Content-Type": "application/json",
  "BM-AUTH-APIKEY": apiKey,
  "BM-AUTH-TIMESTAMP": timestamp,
  "BM-AUTH-SIGNATURE": signature
};
```

### Sending http request

The following code sample sends http request

```
var post_options = {
        host: baseUrl,
        path: path,
        method: 'POST',
        headers: headers
};
var post_req = https.request(post_options, function(res) {
    res.on('data', function (chunk) {
        console.log('Http Response Code: ' + res.statusCode);
        console.log('Response: ' + chunk);
    });
});
post_req.write(data);
post_req.end();
```

## Pagination

API v3 is using cursor-based pagination, allowing users to retrieve a specified number of records before or after a given number.

### Pagination parameters

APIs with support for pagination will return two additional HTTP headers for each request.

- `BM-BEFORE`
- `BM-AFTER`

Then for those APIs, the following query parameters can be passed to handle pagination.

| Name | Mandatory | Description |
|---|---|---|
| limit | no | specifies how many items to return any number between 1 and 200 |
| before | no | return items before given cursor |
| after | no | return items after given cursor |

### Pagination process

Use the following steps:

1. Call the API with no query string params (you can specify the `limit`)
2. Parse the HTTP response and extract the `BM_BEFORE` and `BM-AFTER` http headers
3. If you need to retrieve older records, then pass `before=1234567` as query string with `1234567` being the value of HTTP header `BM-BEFORE`
4. If you need to retrieve newer records, then pass `after=123589` as query string with `123589` being the value of HTTP header `BM-AFTER`
5. Repeat from step 2

## Batch processing

Batch processing is supported for order placement, cancellation, and retrieval APIs, giving more flexibility to traders and market makers for implementing different trading strategies.

batch requests are not considerd atomic actions so each individual item inside the request is handled separately and ensure that an individual response is present for each request.

One use case for batch processing is canceling an existing order and placing a new order with different attributes in a single http call.

Error handling for batch requests is similar to standard REST APIs with exception that some of the error messages are no longer returned as HTTP status.

For instance, when placing a new order, the following two error messages might be returned as a result of invalid request:

| request | http status |
|---|---|
| using price:"12.32abc" | HTTP 400 - invalid due to invalid price |
| placing order with insufficient fund | HTTP 400 - insufficient fund |

However when placing a batch requests of multiple order placements then once the content of the batch is validated then each individual requets return an error message inside the response body hence above errors do nnot result in http 400. Instead you should parse the response body to determine which requests were successful.

Please refer to **Batch Order Placement** section for more information about how to handle the response body and errors for batch requests.

## Order Status

An order can have any of the following statues

- `Accepted`: The order has been accepted by the trading platform
- `Placed`: The order has been submitted to the orderbook, and appropriate funds are locked
- `Partially Matched`: order has been partially matched
- `Fully Matched`:
- `Cancelled`: order has been canceled and no longer in the orderbook
- `Partially Cancelled`: order has been canceled after partially executed
- `Failed`: order was failed by trading engine due to issues like insufficient funds

Some of those statuses are considered final. Hence, the order can no longer be changed.

## Order Types

The following order types are supported

- Limit
- Market
- Stop Limit
- Stop
- Take Profit

## Stop orders

In general, stop orders stay in the orderbook and only are activated once the trigger price of the order is reached by the market. Upon triggering and depending on the type, the order will be placed in the orderbook as Limit or Market order that will then executes as normal.

- For Stop Limit orders, the order type attribute will be "Stop Limit", and an additional field triggerPrice must be supplied.
- For Stop orders, the order type attribute will be "Stop", and an additional field triggerPrice must be supplied with the same number conversion as price. When triggerPrice is reached, a market order will be executed.
- For Take Profit order, the order type attribute will be "Take Profit", and an additional field triggerPrice must be supplied with the same number conversion as price. When triggerPrice is reached, a market order will be executed.
- For both Stop orders and Take Profit orders, depending on existing orders in the orderbook and market price at the time of triggering, they will be executed (similar to Market orders).

## Client Order Id

`clientOrderId` is an optional parameter that can be set by the client applications to track their orders using their internal order management system. clientOrderId is a string field between 1 to 100 with the following characters allowed: `a-z, A-Z, 0-9, -` allowing standard formats like UUID. We have tried to make a distinction between the exchange `orderId` and `clientOrderId` returning both attributes whilst also allowing users to retrieve orders based on both attributes as well, hence APIs like `/v3/orders/{id}` use `id` to demonstrate that either of the orderId or clientOrderId can be passed.

`clientOrderId` is optional for normal order placement. However, it is mandatory if you decide to use batch order placement.

## Time in force

`timeInForce` option allows traders to control order lifetime with the following possible values.

- `GTC`: good till canceled (the default value if not specified)
- `IOC`: immediate or cancel. The order should be canceled immediately after being placed in the orderbook. The order may be matched against existing orders as part of the initial placement in the orderbook, but any volume that is is left from trade matching will be canceled immediately after placement. Therefore possible results of an order with `IOC` option will be either `Fully Matched`, `Partially Matched`, `Cancelled` or `Partially Cancelled`.
- `FOK`: fill or kill. The order is cancelled if it's not executed fully at the time of placement. Possible results of an order with `FOK` option will be either `Fully Matched` or `Cancelled` immediately.

Note When `postOnly` option is set to `true` then the only possible value for `timeInForce` is `GTC` or do not pass any value for timeInForce.

## Desired target amount

This is an optional parameter allowing traders to instruct the system to `sell` (or `buy)` as many number of `instrument` as possible so that the `targetAmount` is reached. For example, as a trader, I'd like to receive $100 (the target amount) by selling as many of my `XRPs` (the actual volume is determined by the system at the time of placing the order). The system then makes the best effort to determine the total volume of `XRP` that is needed to sell in order to generate $100, considering all trading fees, partial order match, etc. This feature eliminates the need for traders to calculate how many XRP's are needed before placing an order, and particularly when the market is moving quickly, this might be a difficult task. This option also works for `Bid` orders with the use case being: I'd like to spend a maximum of $100 on buying `LTC` and the system determines the total volume that I can purchase considering all fees.
`targetAmount` option is only applicable to `Market` orders.

Please note that `price` and `volume` are not required when using `targetAmount`.

## Post only orders

`postOnly` is an optional boolean flag with true/false value. The default is false for all new orders. When this option is set to `true` then it means the order should only be posted to the orderbook if it does not result in any trades at the time of placement. If any part of the order results in trade execution, the order will be canceled. Please note: once the order is accepted as part of the initial placement, then it may execute at any later time depending on the market price movements. The option is only applied for the time of placement. `postOnly` option is only applicable to `Limit` orders. This option is mainly useful for market makers and liquidity providers.

## Self Trade Prevention

`selfTrade` option allows traders to control the possibility of their own orders execute against each other. For instance, if you already have an order already in the orderbook to buy 100 `XRP` for price of 0.50 and then place a new order to sell 100 `XRP` with the same price of 0.50, then the new sell order will be canceled immediately and won't be submitted to the orderbook. The existing order (the buy order for 100 XRP) will continue to stay in the orderbook and will work as normal. The same is true also when the new order has the potential to match existing orders partially. So in above sample (your existing buy order of 100 `XRP`) and if you place a new sell order with selfTrade option to sell 200 `XRP` (given that 100 of those `XRP` will potentially match with an existing order that does not belong to you) then again the entire new sell order will be cancelled immediately due to self trade option. Please also note that this option is only checked at the time of order arrival and only once. If the order stays in the orderbook, then this option is not applicable anymore. Possible values for `selfTrade` option are `A` (self trade is allowed that is the default behavior) and `P` to prevent self trade.

## Market data APIs

### List active markets

Retrieves list of active markets including configuration for each market.

- `baseAssetName`: the asset being purchased or sold. In the case of `ETH-AUD` the base asset is `ETH`
- `quoteAssetName`: the asset that is used to price the base asset. In the case of `ETH_AUD` quote asset is `AUD`
- `marketId`: market id is used across the system
- `minOrderAmount`: minimum amount for an order

- `maxOrderAmount`: maximum amount for an order
- `amountDecimals`: maximum number of decimal places can be used for amounts.
- `priceDecimals`: represents number of decimal places can be used for price when placing orders. For instance for `BTC-AUD` market priceDecimals is 2 meaning that price of `100.12` is valid but `100.123` is not.

## Responses

— 200 OK

GET `/v3/markets`

### Response samples

200

Content type

application/json

Copy Expand all Collapse all

```
[
  - {
        "marketId": "BTC-AUD",
        "baseAsset": "BTC",
        "quoteAsset": "AUD",
        "minOrderAmount": "0.0001",
        "maxOrderAmount": "1000000",
        "amountDecimals": "8",
        "priceDecimals": "2"
    },
  - {
        "marketId": "LTC-AUD",
        "baseAsset": "LTC",
        "quoteAsset": "AUD",
        "minOrderAmount": "0.001",
```

## Get market ticker

Retrieves tikcer for the given marketId.

- `bestBid`: best buy order price
- `bestAsk`: best sell order price
- `lastPrice`: price of the last trade
- `volume24`: represents total trading volume over the past 24 hours for the the given market
- `price24`: price change (difference between the first and last price over 24 hours)
- `low24`: lowest price over the past 24 hours
- `high24`: highest price over the past 24 hours

PATH PARAMETERS

| | |
|---|---|
| `marketId` required | string |

### Responses

— **200** OK

— **404** Market not Found

GET `/v3/markets/{marketId}/ticker`

**Response samples**

**200**

Content type

application/json

Copy Expand all Collapse all

```
{
    "marketId": "BAT-AUD",
    "bestBid": "0.2612",
    "bestAsk": "0.2677",
    "lastPrice": "0.2652",
    "volume24h": "6392.34930418",
    "price24h": "0.0024",
    "low24h": "0.2621",
    "high24h": "0.2708",
    "timestamp": "2019-09-01T10:35:04.940000Z"
}
```

## Get market trades

Retrieves list of most recent trades for the given market. this API supports pagination.

PATH PARAMETERS

| | |
|---|---|
| `marketId` required | string |

QUERY PARAMETERS

| | |
|---|---|
| `before` | integer <int64> Example: `before=78234976` this is part of the pagination parameters. |
| `after` | integer <int64> Example: `after=78234876` this is part of the pagination parameters. |
| `limit` | integer <int32> Example: `limit=10` this is part of the pagination parameters. |

## Responses

— 200 OK

GET /v3/markets/{marketId}/trades

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
[
  - {
        "id": "4107372347",
        "price": "0.265",
        "amount": "11.25",
        "timestamp": "2019-09-02T12:49:42.874000Z",
        "side": "Ask"
    },
  - {
        "id": "4107297908",
        "price": "0.265",
        "amount": "250",
        "timestamp": "2019-09-02T12:15:29.570000Z",
        "side": "Bid"
    }
]
```

## Get market orderbook

Retrieves list of bids and asks for a given market. passing `level=1` returns top 50 for bids and asks. `level=2` returns full orderbook (full orderbook data is cached and usually updated every 10 seconds). Each market order is represented as an array of string `[price, volume]`. The attribute, `snapshotId`, is a uniqueue number associated to orderbook and it changes every time orderbook changes.

PATH PARAMETERS

| | |
|---|---|
| `marketId` required | string |

QUERY PARAMETERS

| | |
|---|---|
| `level` | integer<br>Example: `level=1`<br>specifies the depth of the orderbook. Default level is 1 |

### Responses

— 200 OK

GET `/v3/markets/{marketId}/orderbook`

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
{
    "marketId": "BAT-AUD",
    "snapshotId": 1567334110144000,
  - "asks": [
      + [ … ],
      + [ … ],
      + [ … ],
      + [ … ]
    ],
  - "bids": [
      + [ … ],
      + [ … ]
    ]
}
```

## Get market candles

Retrieves array of candles for a given market. Each candle record is an array of string representing `[time,open,high,low,close,volume]` for the time window specified (default time window is 1 day).

This API can be used to retrieve candles either by pagination (`before`, `after`, `limit`) or by specifying timestamp parameters (`from` and/or `to`). Pagination parameters can't be combined with timestamp parameters and default behavior is pagination when no query param is specified.

When using timestamp parameters as query string, the maximum number of items that can be retrieved is 1000, and depending on the specified timeWindow this can be different time windows. For instance, when using `timeWindow=1d` then up to 1000 days of market candles can be retrieved.

PATH PARAMETERS

| | |
|---|---|
| `marketId` required | string |

QUERY PARAMETERS

| | |
|---|---|
| `timeWindow` | string<br>Example: `timeWindow=1h`<br>values can be `1m`, `1h` and `1d` representing minute, hour and day. Default value is `1d` if not specified |
| `from` | string<br>allows retrieving market candles from a specific time. The value must be timestamp in ISO 8601 format. e.g. `2018-08-20T06:22:11.000000Z` |

| | |
|---|---|
| `to` | string<br>allows retrieving market candles up to a specific time. The value must be timestamp in ISO 8601 format. e.g. `2019-08-20T06:22:11.000000Z` |
| `before` | integer <int64><br>Example: `before=78234976`<br>this is part of the pagination parameters. |
| `after` | integer <int64><br>Example: `after=78234876`<br>this is part of the pagination parameters. |
| `limit` | integer <int32><br>Example: `limit=10`<br>this is part of the pagination parameters. |

## Responses

— 200 OK

GET `/v3/markets/{marketId}/candles`

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
[
    - [
          "2019-09-02T18:00:00.000000Z",
          "15100",
          "15200",
          "15100",
          "15199",
          "4.11970335"
      ],
    - [
          "2019-09-02T17:00:00.000000Z",
          "14879.75",
          "15115",
          "14861.99",
```

## Get multiple tickers

This API works similar to `/v3/markets/{marketId}/ticker` except it retrieves tickers for a given list of marketIds provided via query string (e.g. `?marketId=ETH-BTC&marketId=XRP-BTC`).

To gain better performance, restrict the number of `marketIds` to the items needed for your trading app instead of requesting all markets.

QUERY PARAMETERS

| | |
|---|---|
| `marketId` required | string |

### Responses

— 200 OK

GET `/v3/markets/tickers`

### Response samples

**200**

Content type

application/json

Copy Expand all Collapse all

```
[
  - {
        "marketId": "BTC-AUD",
        "bestBid": "9000",
        "bestAsk": "9900",
        "lastPrice": "8500",
        "volume24h": "1444.44",
        "price24h": "130",
        "low24h": "12",
        "high24h": "50000",
        "timestamp": "2019-07-31T21:32:08.659000Z"
    },
  - {
        "marketId": "LTC-AUD",
        "bestBid": "99.12",
        "bestAsk": "101.14",
        "lastPrice": "100",
        "volume24h": "1199.8",
        "price24h": "10",
        "low24h": "100",
        "high24h": "120",
        "timestamp": "2019-05-02T15:22:51.770000Z"
    }
]
```

## Get multiple orderbooks

This API works similar to `/v3/markets/{marketId}/orderbook` except it retrieves orderbooks for a given list of marketIds provided via query string (e.g. `?marketId=ETH-BTC&marketId=XRP-BTC`).

To gain better performance, restrict the number of `marketIds` to the items needed for your trading app instead of requesting all markets.

Retrieving full orderbook (`level=2`), for multiple markets, was mainly provided for customers who are interested in capturing and keeping full orderbook history. Therefore, it's recommended to call this API with lower frequency as the data size can be large and also cached.

QUERY PARAMETERS

| | |
|---|---|
| `marketId` required | string |

## Responses

— 200 OK

GET `/v3/markets/orderbooks`

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
[
  - {
        "marketId": "BAT-AUD",
        "snapshotId": 1567334110144000,
      + "asks": [ … ],
      + "bids": [ … ]
    },
  - {
        "marketId": "LTC-AUD",
        "snapshotId": 1567334110146000,
      + "asks": [ … ],
      + "bids": [ … ]
    }
]
```

## Order Placement APIs

### Place new order

This API is used to place a new order. Some of the parameteres are mandatory as specified below. The right panel presents the default response with primary attributes. You can also select from the drop down to see an order response with all possible order attributes.

REQUEST BODY SCHEMA: application/json

| | |
|---|---|
| `marketId` required | string<br>specify a marketId e.g. BTC-AUD |
| `price` required | string |
| `amount` required | string |
| `type` required | string<br>Enum: `"Limit"` `"Market"` `"Stop Limit"` `"Stop"` `"Take Profit"`<br>type of the order |
| `side` required | string<br>Enum: `"Bid"` `"Ask"`<br>side of the order |
| `triggerPrice` | string<br>this is mandatory if order type is Stop, Stop Limit or Take Profit |
| `targetAmount` | string |

| | |
|---|---|
| | specifiy target amount when a desired target outcome is required for order execution |
| `timeInForce` | string<br>possible values are GTC (default option) , FOK and IOC |
| `postOnly` | boolean<br>if this is a post-only order |
| `selfTrade` | string<br>A or P |
| `clientOrderId` | string<br>a unique order id speciifed by client app. |

## Responses

> 200 OK

— 400

— 401 Not Authorized

— 403 Forbidden

POST `/v3/orders`

## Request samples

Payload Javascript

Content type
application/json

Copy Expand all Collapse all

```
{
    "marketId": "BTC-AUD",
    "price": "100.12",
    "amount": "1.034",
    "type": "Limit",
    "side": "Bid"
}
```

## Response samples

200 400

Content type

application/json

Example

basic order attributes

Copy Expand all Collapse all

```
{
    "orderId": "7524",
    "marketId": "BTC-AUD",
    "side": "Bid",
    "type": "Limit",
    "creationTime": "2019-08-30T11:08:21.956000Z",
    "price": "100.12",
    "amount": "1.034",
    "openAmount": "1.034",
    "status": "Accepted"
}
```

## List orders

Returns an array of historical orders or open orders only. All query string parametesr are optional so by default and when no query parameter is provided, this API retrieves open orders only for all markets. This API supports pagination only when retrieving all orders `status=all`, When sending using `status=open` all open orders are returned and with no pagination.

QUERY PARAMETERS

| | |
|---|---|
| `marketId` | string |

| | |
|---|---|
| | Example: `marketId=ETH-AUD`<br>by default orders for all markets are returned. specify a marketId for filtering. |
| `before` | integer <int64><br>Example: `before=78234976`<br>this is part of the pagination parameters. |
| `after` | integer <int64><br>Example: `after=78234876`<br>this is part of the pagination parameters. |
| `limit` | integer <int32><br>Example: `limit=50`<br>this is part of the pagination parameters. |
| `status` | string<br>Enum: `"open"` `"all"`<br>returns orders with open status or all statuses. |

## Responses

— 200 OK

GET /v3/orders

## Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
{
    "orderId": "7524",
    "marketId": "BTC-AUD",
    "side": "Bid",
    "type": "Limit",
        "creationTime": "2019-08-30T11:08:21.956000Z",
        "price": "100.12",
        "amount": "1.034",
        "openAmount": "1.034",
        "status": "Accepted"
    }
```

## Cancel open orders

Cancels all open orders for all markets or optionally for a given list of marketIds only.

QUERY PARAMETERS

| | |
|---|---|
| `marketId` | string<br>restricts cancellation for those given marketIds only. can be provided in the form of `marketId=BTC-AUD&marketId=ETH-AUD` |

### Responses

> 200 OK

DELETE `/v3/orders`

### Response samples

200

Content type

application/json

Copy Expand all Collapse all

```
[
    - {
          "orderId": "7524",
          "clientOrderId": "123-456"
      },
    - {
          "orderId": "435",
          "clientOrderId": "abc"
      }
  ]
```

---

## Get an order

Returns an order by using either the exchange `orderId` or `clientOrderId`

PATH PARAMETERS

| | |
|---|---|
| `id` `required` | string |

### Responses

— **200** OK

— **404** Not found

GET `/v3/orders/{id}`

**Response samples**

**200**

Content type
application/json

Copy Expand all Collapse all

```
{
    "orderId": "7524",
    "marketId": "BTC-AUD",
    "side": "Bid",
    "type": "Limit",
    "creationTime": "2019-08-30T11:08:21.956000Z",
    "price": "100.12",
    "amount": "1.034",
    "openAmount": "1.034",
    "status": "Accepted"
}
```

## Cancel an order

Cancels a single order. this API returns http error `400` if the order is already cancelled, matched or partially matched.

PATH PARAMETERS

| | |
|---|---|
| id<br>required | string |

### Responses

> 200 OK

DELETE `/v3/orders/{id}`

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
{
    "orderId": "7524",
    "clientOrderId": "123-456"
}
```

## Replace an Order

This API combines both cancel and placement operations in a single API call.

Upon calling this API, the system attempts to cancel the existing order, and if the cancel operation is successful, then a new order is placed with exactly the same attributes (e.g., type, marketId, etc.) except for `price` and `amount` that are provided as input to this API.

Note:

- The underlying actions of canceling an existing order and placing a new order at this stage are not atomic due to the asynchronous nature of the system. This means while we make the best effort to make sure the existing order is canceled successfully before placing a new order. however, there can be situations where the existing order may still execute and the new order is also placed. (particularly in situations when the market moves quickly in a few milliseconds this can happen).
- For simplicity, this API returns all attribute of the new order including new orderId (and new clientOrderId, if provided)
- This API also returns all possible errors that may occur during normal cancellation and order placement

PATH PARAMETERS

| | |
|---|---|
| `id` required | string |

REQUEST BODY SCHEMA: application/json

| | |
|---|---|
| `price` required | string |
| `amount` required | string |

| | |
|---|---|
| `clientOrderId` | string<br>optional. a unique order id speciifed by client app. |

## Responses

> 200 OK

— 400

PUT `/v3/orders/{id}`

### Request samples

**Payload**

Content type
application/json

Copy Expand all Collapse all

```
{
    "price": "100.12",
    "amount": "1.034"
}
```

### Response samples

**200** **400**

Content type
application/json

Example
basic order attributes

Copy Expand all Collapse all

```
{
        "orderId": "7524",
        "marketId": "BTC-AUD",
        "side": "Bid",
        "type": "Limit",
        "creationTime": "2019-08-30T11:08:21.956000Z",
        "price": "100.12",
        "amount": "1.034",
        "openAmount": "1.034",
        "status": "Accepted"
    }
```

## Batch Order APIs

### Place and Cancel orders

Use this API to place multiple new orders or cancel existing ones via a single request. The request for batch processing is an array of items, and each item contains instructions for an order placement and a cancellation. There are restrictions on the number of items in a batch (currently set to 10) so a batch can contain up to 4 items in any form that is needed. For instance it can contain four placements (hence four items in the array) or four cancellations or array of two items with two order placements and two cancellations.

Batch operations are only containers for multiple requests, so each individual request is handled separately from the rest of the requests in the batch.

Once all items in the batch are processed then a single response containing orders added and orders cancalled is returned along with an attribute called `unprocessedRequests` that is an array of any item in the batch thet can't be processed.

One major difference between processing batch requests and individual ones is error handling. In the case of processing an individual order placement, if the request is invalid (e.g. invalid price), then a HTTP 400 error is returned. However, with batch processing (even with single item) you will receive an HTTP code of 200, and you should parse the `unprocessedRequests` inside the response body to determine if there are any errors processing individual requests.

There are cases where the entire batch request is considered invalid and an HTTP error `400` is returned (e.g., batch size exceeds the limit) so when handling batch requests, your client needs to handle HTTP status codes as well as error messages inside the response body.

Another difference is that you must provide `clientOrderId` when placing orders in batch. This allows items inside a batch request to be tracked and processed accurately. When cancelling orders, you can either use `orderId` or `clientOrderId` within the request. `clientOrderId` is only mandatory for creating new orders.

`requestId` that appears inside the `unprocessedRequests represents whatever id was used to identify an order (e.g., clientOrderId or orderId)

One sample use case of batch processing is simulating update order in a single http call that is cancelling an existing order and then creating a new order. In order to achieve this outcome, use the following request sample:

```
[{"cancelOrder":{"clientOrderId":"27"}},{"placeOrder":{"marketId":"XRP-AUD","side"
```

In above sample, the existing order with id `27` is cancelled and a new order with id 28 will be created.

You can still take advantage of `/3/batchorders{ids}` in order to cancel orders and use this API for placing orders only.

REQUEST BODY SCHEMA: application/json

Array [

| | |
|---|---|
| placeOrder › | object |
| cancelOrder › | object |

]

## Responses

› 200 OK

— 400 Bad Request

POST /v3/batchorders

## Request samples

**Payload**

Content type
application/json

Copy Expand all Collapse all

```
[
  - {
      + "placeOrder": { … },
      + "cancelOrder": { … }
    },
  - {
      + "placeOrder": { … },
      + "cancelOrder": { … }
    }
]
```

## Response samples

**200** **400**

Content type
application/json

Example
basic order attributes

Copy Expand all Collapse all

```
{
  - "placeOrders": [
      + { … }
    ],
  - "cancelOrders": [
      + { … }
    ],
  - "unprocessedRequests": [
      + { … },
      + { … }
    ]
}
```

## Get orders by Id

Retrieves batch of orders by using either the exchange `orderId` or `clientOrderId`. You can specify a comma separated list of ids `/v3/batchorders/abc,dbc,3a,4`

PATH PARAMETERS

| | |
|---|---|
| `ids` required | string<br>comma delimited list of ids |

### Responses

— **200** OK

— **400** Bad Request

GET `/v3/batchorders/{ids}`

### Response samples

200 400

Content type
application/json

Copy Expand all Collapse all

```
{
  - "orders": [
      + { … }
    ],
  - "unprocessedRequests": [
      + { … },
      + { … }
    ]
```

## Cancel orders by Id

This API can be used to cancel a list of orders specified by id in a single request e.g. `/v3/batchorders/6,7,1`

PATH PARAMETERS

| | |
|---|---|
| `ids` required | string<br>comma delimited list of ids |

### Responses

— 200 OK

DELETE `/v3/batchorders/{ids}`

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
{
  - "cancelOrders": [
      + { … },
      + { … }
    ],
  - "unprocessedRequests": [
      + { … }
    ]
}
```

## Trade APIs

### List trades

Retrieves trades and optionally filters by marketId or orderId/clientOrderId. The default behavior, when no query parameter is specified, is to return your most recent trades for all orders and markets. When a valid order id is provided then all trades for the order is returned. provding `marketId` also filters trades. Mixing `orderId` and `marketId` parameters is not supported.

QUERY PARAMETERS

| | |
|---|---|
| `marketId` | string<br>optionally filter trades by marketId (e.g. XRP-AUD) |
| `orderId` | string<br>optionally list all trades for a single order |
| `before` | integer <int64><br>Example: `before=78234976`<br>this is part of the pagination parameters. |
| `after` | integer <int64><br>Example: `after=78234876`<br>this is part of the pagination parameters. |
| `limit` | integer <int32><br>Example: `limit=10`<br>this is part of the pagination parameters. |

## Responses

— 200 OK

GET /v3/trades

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
[
 - {
        "id": "36014819",
        "marketId": "XRP-AUD",
        "timestamp": "2019-06-25T16:01:02.977000Z",
        "price": "0.67",
        "amount": "1.50533262",
        "side": "Ask",
        "fee": "0.00857285",
        "orderId": "3648306",
        "liquidityType": "Taker",
        "clientOrderId": "48"
    },
  - {
        "id": "3568960",
        "marketId": "GNT-AUD",
        "timestamp": "2019-06-20T08:44:04.488000Z",
        "price": "0.1362",
        "amount": "0.85",
        "side": "Bid",
        "fee": "0.00098404",
        "orderId": "3543015",
        "liquidityType": "Maker"
```

## Get trade by id

Retrieves a trade by id

PATH PARAMETERS

| | |
|---|---|
| id required | string |

### Responses

— 200 OK

```
GET /v3/trades/{id}
```

# Fund Management APIs

## Request to withdraw

This API is used to request to withdraw of crypto assets or `AUD`.

## withdrawal/deposit Status

- `Accepted`: The withdrawal request has been accepted for processing
- `Pending Authorization`: The withdrawal request has been accepted and is being processed
- `Complete`: The withdrawal/deposit request has been successfully processed
- `Cancelled`: The withdrawal request has been cancelled due to issues with the request
- `Failed`: The withdrawal request has been failed due to a system error

REQUEST BODY SCHEMA: application/json

| | |
|---|---|
| `assetName` `required` | string<br>name of the asset to withdraw e.g. `AUD` or `BTC` |
| `amount` `required` | string<br>amount to withdraw |
| `toAddress` | string<br>destination address for crypto withdrwawal. mandatory for crypto assets. For `XRP` withdrawals, you can optionally add destination tag to the address with `?dt=12345` e.g. `1EJKj147QmEzywLnLpuxSr6SoPr1p62VBX?dt=123456` |
| `accountName` | string<br>optional for AUD withdrawal. when not speciifed default bank information is used |
| `accountNumber` | string<br>optional for AUD withdrawal. when not speciifed default bank information is used |
| `bsbNumber` | string<br>optional for AUD withdrawal. when not speciifed default bank information is used |
| `bankName` | string<br>optional for AUD withdrawal. when not speciifed default bank information is used |

## Responses

— 200 OK

— 400

POST `/v3/withdrawals`

### Request samples

Payload

Content type
application/json

Example
crypto

Copy Expand all Collapse all

```
{
    "assetName": "XRP",
    "amount": "25",
    "toAddress": "abc"
}
```

### Response samples

200 400

Content type
application/json

Example
crypto withdraw example

Copy Expand all Collapse all

```
{
    "id": "4126657",
    "assetName": "XRP",
    "amount": "25",
    "type": "Withdraw",
    "creationTime": "2019-09-04T00:04:10.973000Z",
    "status": "Pending Authorization",
    "description": "XRP withdraw from [me@test.com] to Address: abc amount: 25 fee: 0",
    "fee": "0",
    "lastUpdate": "2019-09-04T00:04:11.018000Z"
}
```

## List withdrawals

Returns list of withdrawals. This API supports pagination

QUERY PARAMETERS

| | |
|---|---|
| `before` | integer <int64> Example: `before=78234976` this is part of the pagination parameters. |
| `after` | integer <int64> Example: `after=78234876` this is part of the pagination parameters. |
| `limit` | integer <int32> Example: `limit=10` this is part of the pagination parameters. |

### Responses

— **200** OK

GET `/v3/withdrawals`

**Response samples**

200

Content type
application/json

Copy Expand all Collapse all

```
[
  - {
        "id": "123989",
        "assetName": "BTC",
        "amount": "0.3",
        "type": "Withdraw",
        "creationTime": "2019-08-27T21:41:56.832000Z",
        "status": "Pending Authorization",
        "description": "BTC withdraw from [me@test.io] to Address: 3QJsRCW3qSinyC amount:
        "fee": "0",
        "lastUpdate": "2019-08-27T21:41:57.004000Z",
      + "paymentDetail": { … }
    },
  - {
        "id": "1167870",
        "assetName": "AUD",
        "amount": "0.15710206",
        "type": "Deposit",
        "creationTime": "2019-08-16T23:23:39.452000Z",
        "status": "Complete",
        "description": "EFT Deposit, $ 0.15710206",
        "fee": "0",
        "lastUpdate": "2019-08-16T23:23:39.603000Z"
    }
]
```

## Get withdraw by Id

This API is used to request to get withdraw by id.

PATH PARAMETERS

| | |
|---|---|
| `id` required | string |

### Responses

— **200** OK

— **404** Not found

GET `/v3/withdrawals/{id}`

### Response samples

**200**

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "4126657",
    "assetName": "XRP",
    "amount": "25",
    "type": "Withdraw",
    "creationTime": "2019-09-04T00:04:10.973000Z",
    "status": "Pending Authorization",
    "description": "XRP withdraw from [me@test.com] to Address: abc amount: 25 fee: 0",
    "fee": "0",
    "lastUpdate": "2019-09-04T00:04:11.018000Z"
}
```

## List deposits

Returns list of depoists. This API supports pagination

QUERY PARAMETERS

| | |
|---|---|
| `before` | integer <int64><br>Example: `before=78234976`<br>this is part of the pagination parameters. |
| `after` | integer <int64><br>Example: `after=78234876`<br>this is part of the pagination parameters. |
| `limit` | integer <int32><br>Example: `limit=10`<br>this is part of the pagination parameters. |

## Responses

— **200** OK

GET `/v3/deposits`

### Response samples

**200**

Content type

application/json

Copy Expand all Collapse all

```
[
  - {
        "id": "123989",
        "assetName": "BTC",
        "amount": "0.3",
        "type": "Withdraw",
        "creationTime": "2019-08-27T21:41:56.832000Z",
        "status": "Pending Authorization",
        "description": "BTC withdraw from [me@test.io] to Address: 3QJsRCW3qSinyC amount:
        "fee": "0",
        "lastUpdate": "2019-08-27T21:41:57.004000Z",
      + "paymentDetail": { … }
    },
  - {
        "id": "1167870",
        "assetName": "AUD",
        "amount": "0.15710206",
        "type": "Deposit",
        "creationTime": "2019-08-16T23:23:39.452000Z",
        "status": "Complete",
        "description": "EFT Deposit, $ 0.15710206",
        "fee": "0",
        "lastUpdate": "2019-08-16T23:23:39.603000Z"
    }
]
```

## Get deposit by Id

This API returns a deposit by id.

PATH PARAMETERS

| | |
|---|---|
| `id` required | string |

### Responses

— **200** OK

— **404** Not found

GET /v3/deposits/{id}

**Response samples**

**200**

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "17866",
    "assetName": "BTC",
    "amount": "0.15710206",
    "type": "Deposit",
    "creationTime": "2019-08-16T23:19:03.553000Z",
    "status": "Complete",
    "description": "BITCOIN Deposit, B 0.15710206",
    "fee": "0",
    "lastUpdate": "2019-08-16T23:19:03.619000Z",
  - "paymentDetail": {
        "txId": "E1264A7D5742480B28494"
    }
}
```

## List deposits/withdrawals

A transfer record refers either to a deposit or withdraw and this API returns list of transfers covering both depoists and withdrawals. This API supports pagination

QUERY PARAMETERS

| | |
|---|---|
| `before` | integer <int64> Example: `before=78234976` this is part of the pagination parameters. |
| `after` | integer <int64> Example: `after=78234876` this is part of the pagination parameters. |
| `limit` | integer <int32> Example: `limit=10` this is part of the pagination parameters. |

## Responses

— **200** OK

GET `/v3/transfers`

### Response samples

**200**

Content type
application/json

Copy Expand all Collapse all

```
[
  - {
        "id": "123989",
        "assetName": "BTC",
        "amount": "0.3",
        "type": "Withdraw",
        "creationTime": "2019-08-27T21:41:56.832000Z",
        "status": "Pending Authorization",
        "description": "BTC withdraw from [me@test.io] to Address: 3QJsRCW3qSinyC amount:
        "fee": "0",
        "lastUpdate": "2019-08-27T21:41:57.004000Z",
      + "paymentDetail": { … }
    },
  - {
        "id": "1167870",
        "assetName": "AUD",
        "amount": "0.15710206",
        "type": "Deposit",
        "creationTime": "2019-08-16T23:23:39.452000Z",
        "status": "Complete",
        "description": "EFT Deposit, $ 0.15710206",
        "fee": "0",
        "lastUpdate": "2019-08-16T23:23:39.603000Z"
    }
]
```

## Get deposits/withdrawals by Id

This API retruns either deposit or withdrawal by id

### Responses

— **200** OK

— **404** Not found

GET /v3/transfers/{id}

## Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "17866",
    "assetName": "BTC",
    "amount": "0.15710206",
    "type": "Deposit",
    "creationTime": "2019-08-16T23:19:03.553000Z",
    "status": "Complete",
    "description": "BITCOIN Deposit, B 0.15710206",
    "fee": "0",
    "lastUpdate": "2019-08-16T23:19:03.619000Z",
  - "paymentDetail": {
        "txId": "E1264A7D5742480B28494"
    }
}
```

## Get deposit address

returns deposit address for the given asset

QUERY PARAMETERS

| | |
|---|---|
| `assetName` required | string<br>asset name for the deposit address |

### Responses

— **200** OK

GET `/v3/addresses`

### Response samples

**200**

Content type
application/json

Copy Expand all Collapse all

```
{
    "address": "1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2",
    "assetName": "BTC"
}
```

## get withdrawal fees

Returns fees associated with withdrawals. This API is public and does not require authentication as the fees as system wide and published on the website

### Responses

— **200** OK

GET `/v3/withdrawal-fees`

### Response samples

200

Content type

application/json

Copy Expand all Collapse all

```
[
  - {
        "assetName": "AUD",
        "fee": "0"
    },
  - {
        "assetName": "BTC",
        "fee": "0.0003"
    },
  - {
        "assetName": "BCHABC",
        "fee": "0.001"
    }
]
```

## List assets

Retrieves list of assets including configuration

- `assetName`: name of the asset that us used for trading or investment
- `minDepositAmount`: minimum amount to deposit
- `maxDepositAmount`: maximum amount to deposit
- `depositFee`: deposit fee
- `depositDecimals`: number of decimal places allowed for deposits
- `minWithdrawalAmount`: minimum amount to withdraw
- `maxWithdrawalAmount`: maximum amount to withdraw
- `withdrawalFee`: withdrawal fee
- `withdrawalDecimals`: number of decimal places allowed for withdrawals

## Responses

— **200** OK

GET `/v3/assets`

### Response samples

**200**

Content type

application/json

Copy Expand all Collapse all

```
[
  - {
        "assetName": "BTC",
        "minDepositAmount": "0.0001",
        "maxDepositAmount": "1000000",
        "depositDecimals": "8",
        "minWithdrawalAmount": "0.0001",
        "maxWithdrawalAmount": "1000000",
        "WithdrawalDecimals": "8",
        "withdrawalFee": "0",
        "depositFee": "0"
    }
]
```

# Account APIs

## Get trading fees

Returns 30 day trading fee volume plus trading fee per market covering `marker` and `taker`.

### Responses

**— 200** OK

```
GET /v3/accounts/me/trading-fees
```

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
{
    "volume30Day": "0.0098275",
  - "feeByMarkets": [
      + { … },
      + { … },
      + { … }
    ]
}
```

## Get withdrawal limits

This API is used

```
GET /v3/accounts/me/withdrawal-limits
```

## Get balances

Returns list of assets covering balance, available, and locked amount for each asset due to open orders or pending withdrawals. This formula represents the relationship between those three elements: `balance = available + locked`

### Responses

— **200** OK

```
GET /v3/accounts/me/balances
```

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
[
  - {
        "assetName": "LTC",
        "balance": "5",
        "available": "5",
        "locked": "0"
    },
  - {
        "assetName": "ETH",
        "balance": "1.07583642",
        "available": "1.0",
        "locked": "0.07583642"
    }
]
```

## Get transactions

Returns detail ledger recoerds for underlying wallets. This API supports pagination.

QUERY PARAMETERS

| | |
|---|---|
| `assetName` | string<br>Example: `assetName=BTC`<br>filter transactions for specific asset |
| `before` | integer <int64><br>Example: `before=78234976`<br>this is part of the pagination parameters. |
| `after` | integer <int64><br>Example: `after=78234876`<br>this is part of the pagination parameters. |
| `limit` | integer <int32><br>Example: `limit=10`<br>this is part of the pagination parameters. |

## Responses

**— 200** OK

GET `/v3/accounts/me/transactions`

### Response samples

**200**

Content type

application/json

Copy Expand all Collapse all

```
[
 - {
        "id": "1759",
        "creationTime": "2015-02-21T21:49:54.911000Z",
        "description": "Sell 0.3000BTC @ AUD 200.0000 Trading fee",
        "assetName": "AUD",
        "amount": "0.5082",
        "balance": "81.9401",
        "type": "Trading Fee",
        "recordType": "Trade",
        "referenceId": "17949"
    },
- {
      "id": "17958",
      "creationTime": "2015-02-21T21:49:54.906000Z",
      "description": "Sell 0.3000BTC @ AUD 200.0000 Trade settled",
      "assetName": "AUD",
      "amount": "60",
      "balance": "82.4483",
      "type": "Sell Order",
      "recordType": "Trade",
      "referenceId": "17949"
  },
- {
      "id": "15160",
      "creationTime": "2014-11-12T11:30:02.773000Z",
      "description": "Sell 0.1000BTC @ AUD 4.5100 Fully matched at 4.5100",
      "assetName": "BTC",
      "amount": "0.1",
      "balance": "99.144778",
      "type": "Sell Order",
      "recordType": "Trade",
      "referenceId": "14435"
  },
- {
      "id": "8809",
      "creationTime": "2014-10-05T21:18:19.714000Z",
      "description": "XRP deposit was successfull.Reference: 1",
      "assetName": "XRP",
      "amount": "100",
      "balance": "100",
      "type": "Deposit",
```

## Report APIs

### Create new report

request to generate a new report.

REQUEST BODY SCHEMA: application/json

| | |
|---|---|
| `type` required | string<br>type of the report. the only accepted value is `TransactionReport` at this stage. |
| `format` required | string<br>value can be either `csv` or `json` |

POST `/v3/reports`

### Request samples

**Payload**

Content type
application/json

Copy Expand all Collapse all

```
{
    "type": "TransactionReport",
    "format": "json"
}
```

## Get report by id

This API returns details of the report once it's been created via the previous API.

On average report generation takes about 20 seconds so please allow at least 10 seconds and recommended 30 seconds before attempting to get detail of the report after requesting it via the previous API. Trying too quickly to get detail a newly created report will result in http `404` response. A successful response of this API contains a link that you can use to download the report content.

The attribute `contentUrl` inside the above response is a link to download the report content (in either `json` or `csv` format). Please note that report content files are only available for download for up to 30 minutes after creation time.

Transaction report covers all historical changes made to all of your wallets including deposit/withdrawals, order executions and trading fees.

sample report content in json format:

```
[{"transactionId":"12345","creationTime":"2017-12-11T06:01:28Z","recordType":"Fund
 {"transactionId":"830309","creationTime":"2018-07-11T06:26:34Z","recordType":"Tra
 {"transactionId":"830310","creationTime":"2018-07-11T06:26:34Z","recordType":"Tra
 {"transactionId":"830311","creationTime":"2018-07-11T06:26:34Z","recordType":"Tra
```

- `referenceId` in this report represents either orderId or id of the fund transfer.
- For orders, multiple transactions will have the same referenceId that covers increase/decrease amounts in corresponding wallets (e.g. buy BTC using AUD wallet) and also fees paid.
- `recordType` can be either "Fund Transfer" (representing deposits and withdrawals) or "Trade" (representing buys, sells, and trading fees)
- `balance` represents the balance after each transaction

PATH PARAMETERS

| | |
|---|---|
| `reportId` required | string |

## Responses

— 200 OK

GET `/v3/reports/{id}`

### Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "jsqmkd72lmd13cd0",
    "contentUrl": "https://report.s3.ap-southeast-2.amazonaws.com/jsqmkd72lmd13cd0",
    "creationTime": "2019-08-20T18:08:06.110000Z",
    "type": "TransactionReport",
    "status": "Complete",
    "format": "json"
}
```

## Misc APIs

### Get server time

Returns the server time.

#### Responses

— **200** OK

GET `/v3/time`

#### Response samples

**200**

Content type

application/json

Copy Expand all Collapse all

```
{
    "timestamp": "2019-09-01T18:34:27.045000Z"
}
```

## List of API error codes

Below covers all error codes returned by API

| Error Code | Description | Category |
|---|---|---|
| InvalidPrice | invalid order price | orders |
| InvalidAmount | invalid order amount | orders |
| InvalidTriggerPrice | invalid trigger price | orders |
| InvalidTargetAmount | | orders |
| InvalidOrderSide | | orders |
| InvalidOrderType | | orders |
| TradingNotAvailable | | orders |
| TradingNotAvailableForMarket | | orders |
| OrderTypeNotAvailable | | orders |
| InvalidTimeInForceOption | | orders |
| InvalidSelfTradeOption | | orders |
| InvalidOrderId | | orders |
| OrderNotFound | | orders |
| DuplicateClientOrderId | | orders |
| InvalidClientOrderId | | orders |
| InsufficientFund | | orders |
| OrderAlreadyCancelled | | orders |

| Error Code | Description | Category |
|---|---|---|
| OrderBeingCancelled | | orders |
| OrderStatusIsFinal | | orders |
| TradeNotFound | | orders |
| IncorrectDecimalPoints | | orders |
| FeatureNotAvailableForMarket | | orders |
| InvalidAPIKey | | auth |
| InvalidAuthTimestamp | | auth |
| InvalidAuthSignature | | auth |
| InsufficientAPIPermission | | auth |
| InvalidMarketId | market symbol is invalid | common |
| InvalidAccount | something is wrong with accont setup | common |
| InvalidBatchRequest | | common |
| InvalidParameterCombination | | common |
| DuplicateIdInRequest | | common |
| InvalidIdParameter | | common |
| InvalidPaginationParameter | | common |
| InvalidFeatureCombination | | common |
| MissingArgument | | common |
| FeatureNotAvailable | | common |
| MissingArgument | | common |
| InvalidAssetName | | common |
| AmountExceedAvailableFund | | fund |
| AmountIsTooLow | | fund |
| WithdrawIsNotAvailable | | fund |
| TransferNotFound | | fund |
| DepositAddressNotAvailable | | fund |
| InvalidAddress | | fund |

| Error Code | Description | Category |
|---|---|---|
| MarketNotFound | | market |
| InvalidTimeWindow | | market |
| InvalidTimestamp | | market |
| InvalidOrderbookLevel | | market |
| ReportNotFound | | report |
| InvalidReportParameter | | report |
| BadRequest | | http status |
| UnAuthorized | | http status |
| Forbidden | | http status |
| NotFound | | http status |
| TooManyRequests | | http status |
| InternalServerError | | http status |
| BadGateway | | http status |

# WebSocket Overview

BTC Markets' WebSocket feed provides real-time market data covering orderbook updates, order life cycle and trades.

In order to start receiving real time messages, a WebSocket connection needs to be made followed by a message to subscribe to channels and also marketIds you are interested in.

### Endpoint

The endpoint for WebSocket v2 is: `wss://socket.btcmarkets.net/v2`

### Channels

`tick`, `trade`, `orderbook`, `orderbookUpdate`, `orderChange`, `fundChange`, `heartbeat`

### MarketIds

Market ids represent a market and are used in order to filter events for only specific markets you are interested in.

You should be able to get list of active markets from here: https://api.btcmarkets.net/v3/markets

Format: `BTC-AUD`, `XRP-BTC`, etc.

Note: marketIds are only applicable to public events (e.g. `tick`, `orderbook`, `trade`)

### Message types

All messages published include a `json` attribute called `messageType` that represents type of event that is being received.

Those message types events include:

- `tick`
- `trade`
- `orderbook`
- `orderbookUpdate`
- `orderChange`
- `fundChange`
- `error`
- `heartbeat`

### Subscriptions

Sending `subscribe` message allows you to start receiving events for the specified channels and marketIds. If you need to change subscription then simply send a new `subscribe` message with new channel names and marketIds.

### Managing subscriptions

Whilst sending `subscribe` message works in most situations, however you may want to have the flexibility to add or remove subscriptions instead of subscribing to all channels/markets at the same time.

In those situations send the same subscription message (all rules applies for authentication, marketIds, etc) and the message type will be `addSubscription` or `removeSubscription`. If you need to remove subscription for all markets fo a given channel, just send empty list of marketIds.

Notes when using `addSubscription` and `removeSubscription`

- In order to use those messages, you will need an existing subscription so make sure to send a valid subscription message ast least once.
- With your subscription message (and also add/remove messages) send additional parameter: `clientType: "api"`.

### Client libraries

Below are working example of how to connect to this WebSocket feed in different languages:

- Javascript: https://github.com/ngin-io/websocket-client-node
- Java: https://github.com/ngin-io/websocket-client-java
- Python: https://github.com/ngin-io/websocket-client-python

### Rate limit

New connections to the WebSocket feed are rate limited to 3 attempts per 10 seconds per IP. Frequent WebSocket connections that exceed the rate limit will be closed by the server.

### Connection issues

From time to time your WebSocket connection may be disconnected (e.g. as we upgrade software on our servers). We recommend adding logic to your client in order to refresh your connection every 24 hours or in case if the connection drops out.

## WebSocket Public Events

## Tick event

The `tick` event is published every time `lastPrice`, `bestBid` or `bestAsk` is updated for a market which is the result of orderbook changes or trade matches.

sample event:

```
  { marketId: 'BTC-AUD',
    timestamp: '2019-04-08T18:56:17.405Z',
    bestBid: '7309.12',
    bestAsk: '7326.88',
    lastPrice: '7316.81',
    volume24h: '299.12936654',
    messageType: 'tick'
  }
```

## Trade event

In order to receive `trade` events please add `trade` to the list of channels when subscribing via WebSocket.

sample event:

```
{ marketId: 'BTC-AUD',
  timestamp: '2019-04-08T20:54:27.632Z',
  tradeId: 3153171493,
  price: '7370.11',
  volume: '0.10901605',
  side: 'Ask',
  messageType: 'trade'
}
```

## Orderbook event

In order to receive `orderbook` events please add `orderbook` to the list of channels when subscribing via WebSocket. The current orderbook event represents the latest orderbook state and maximum 50 bids and asks are included in each event.

For efficiency the `bids` and `asks` are are published as arrays of `tuples` representing `[price, volume]` each order in the orderbook.

sample event:

```
{ marketId: 'BTC-AUD',
  timestamp: '2019-04-08T22:23:37.643Z',
  bids:
   [ [ '7418.46', '0.04' ],
     [ '7418.45', '0.56' ],
     [ '7100', '0.01' ] ]
    asks:
     [ [ '7437.53', '0.76' ],
       [ '7437.54', '0.3646349' ],
       [ '7446.94', '0.6' ],
       [ '7700', '0.1' ] ]
    messageType: 'orderbook'
}
```

## OrderbookUpdate event

In many cases, it's more appropriate to maintain a local copy of the exchange orderbook by receiving only updates instead of the entire orderbook.

Some advantages of using `orderbookUpdate` vs `orderbook` are:

1. `orderbook` event only delivers top 50 orders whereas `orderbookUpdate` covers the entire orderbook
2. `orderbook` uses larger bandwidth due to the size of the messages always that includes 50 bids/asks, whereas `orderbookUpdate` only delivers small incremental changes
3. We make the best effort to deliver all individual updates to the orderbook via `orderbookUpdate` message. However, `orderbook` may send a single message if two updates happen at a very close millisecond interval

### Orderbook snapshot message:

Subscribing to `orderbookUpdate` channel allows you to receive a snapshot of the orderbook at first (immediately after subscription), and then the subsequent messages only provide updates to the orderbook. The initial orderbook snapshot message covers all bids/asks represented as arrays of [price, volume, count] tuples as well as `snapshot:true` attribute.

Sample snapshot message:

```
{ marketId: 'LTC-AUD',
  snapshot: true,
  timestamp: '2020-01-08T19:47:13.986Z',
  snapshotId: 1578512833978000,
    bids:
    [ [ '99.57', '0.55', 1 ],
      [ '97.62', '3.20', 2 ],
      [ '97.07', '0.9', 1 ],
      [ '96.7', '1.9', 1 ],
      [ '95.8', '7.0', 1 ] ],
    asks:
      [ [ '100', '3.79', 3 ],
        [ '101', '6.32', 2 ] ],
    messageType: 'orderbookUpdate'
}
```

### Orderbook update message:

The message format for subsequent orderbook updates is the same as the initial snapshot except that it does not have the element `snapshot:true` in it. The subsequent messages only provide updates to the orderbook. The update message is in the form of an array of [price, volume, count] tuples aggregating order volumes for a given price and total number of orders for that price. For instance, ["99.57","0.55",2] means the total volume of bids for the price of `99.57` is `0.55`, and there is a single order with that price.

Sample update message:

```
{ marketId: 'LTC-AUD',
  timestamp: '"2020-01-08T19:47:24.054Z',
  snapshotId: 1578512844045000,
  bids:  [ ['99.81', '1.2', 1 ], ['95.8', '0', 0 ]],
  asks: [ ['100', '3.2', 2 ] ],
  messageType: 'orderbookUpdate'
}
```

### How to use orderbook updates:

In above sample update message:

- A new bid order has been added to the orderbook with price of `99.81` and total volume of `1.2`
- The bid order with price of `95.8` has been removed from the orderbook (hence both the volume and count are zero)
- The total volume and order count for ask order with price of 100 has been changed. New total volume is `3.2` and 2 orders with that price

Assuming you have a local copy of the exchange orderbook (as per sample snapshot above) then applying orderbook update means:

- Add a new bid order tuple with price of `99.81` to your orderbook copy
- Remove the bid order tuple with price of `95.8` from your orderbook copy
- Replace the ask order tuple with price of `100` with new volume and count in your orderbook copy

### Maintaining a local copy of the exchange orderbook

The following steps can be used to maintain an up-to-date copy of the exchange orderbook for a given market:

- Subscribe to `orderbookUpdate` channel and provide a marketId
- Start to queue all incoming messages
- As soon as you receive a message with `snapshot:true` then keep it as your local orderbook copy
- Iterate through the items in the queue and apply changes to your local orderbook copy
- When applying items from the queue, disregard items with snapshotId smaller or equal to the snapshotId of your local orderbook copy

- Once all items in the queue are processed then your local orderbook copy is up to date
- Now apply all incoming messages as they arrive

It's a good practice to refresh your copy at some interval (e.g. once every few hours)

## Heartbeat event

if you subscribe to `heartbeat` event then the server will send you a heartbeat event every 5 seconds.

heartbeat event: `javascript { messageType: 'heartbeat', channels: [ { name: 'orderChange' }, { name: 'orderbook', marketIds: [ 'BTC-AUD', 'XRP-AUD' ] }, { name: 'heartbeat' } ] }` Note: Once a new subscription request is confirmed, a single `heartbeat` event is published to the client in order to confirm the connection working. This is regardless of requesting to subscribe to `heartbeat` channel.

## Error event

In case of errors, a message type of `error` is published.

- Authentication error
- Invalid input error
- Internal server error
- Throttle error

  sample error events:

- Invalid Channel names

```
{
  messageType: 'error',
  code: 3,
  message: 'invalid channel names'
}
```

- Invalid MarketIds

```
{ messageType: 'error',
code: 3,
message: 'invalid marketIds'
}
```

- Authentication error

```
{ messageType: 'error',
code: 1,
message: 'authentication failed. invalid key'
}
```

## Sample code (javascript)

Below sample code connects to `tick` channel. The same code can be used to subscribe to other public channels including heartbeat.

```
const WebSocket = require('ws');
const ws = new WebSocket('wss://socket.btcmarkets.net/v2');

var request = {
  marketIds:marketIds,
  channels: channels,
  messageType: 'subscribe'
}

ws.on('open', function open() {
  ws.send(JSON.stringify(request));
});

ws.on('message', function incoming(data) {
  console.log(data);
});
```

# WebSocket Private Events

## Authenticated events

Receiving events about life cycle of your orders require sending authentication information when subscribing to events. The authentication is similar to public trading API authentication using your API key and secret to sign the message when subscribing via WebSocket.

Below is sample `javascript` code with authentication:

```
const crypto = require('crypto');
const WebSocket = require('ws');

const key = 'your api key';
const secret = 'your api key secret';

const ws = new WebSocket('wss://socket.btcmarkets.net/v2');

const now = Date.now();
const strToSign =  "/users/self/subscribe" + "\n" + now;
const signature = signMessage(secret, strToSign);

var request = {
  marketIds:['BTC-AUD'],
  channels: ['orderChange', 'heartbeat'],
  key: key,
  signature: signature,
  timestamp: now,
  messageType: 'subscribe'
}

ws.on('open', function open() {
  ws.send(JSON.stringify(request));
});

ws.on('message', function incoming(data) {
  console.log(data);
});

function signMessage(secret, message) {
  var key = Buffer.from(secret, 'base64');
  var hmac = crypto.createHmac('sha512', key);
  var signature = hmac.update(message).digest('base64');
```

```
    return signature;
  }
```

## Order life cycle events

Below are events that are published for every step of order processing.

### Placed

```
  { orderId: 79003,
    marketId: 'BTC-AUD',
    side: 'Bid',
    type: 'Limit',
    openVolume: '1',
    status: 'Placed',
    triggerStatus: '',
    trades: [],
    timestamp: '2019-04-08T20:41:19.339Z',
    messageType: 'orderChange'
  }
```

### Fully Matched

```
  { orderId: 79033,
    marketId: 'BTC-AUD',
    side: 'Bid',
    type: 'Limit',
    openVolume: '0',
    status: 'Fully Matched',
    triggerStatus: '',
    trades: [{
               tradeId:31727,
               price":'0.1634',
               volume":'10',
               fee:'0.001',
               liquidityType:'Taker'
             }],
    timestamp: '2019-04-08T20:50:39.658Z',
    messageType: 'orderChange'
  }
```

### Cancelled

```
 { orderId: 79003,
   marketId: 'BTC-AUD',
   side: 'Bid',
   type: 'Limit',
   openVolume: '1',
   status: 'Cancelled',
   triggerStatus: '',
   trades: [],
   timestamp: '2019-04-08T20:41:41.857Z',
   messageType: 'orderChange'
 }
```

### Partially Matched

```
 { orderId: 79003,
   marketId: 'BTC-AUD',
   side: 'Bid',
   type: 'Limit',
   openVolume: '1',
   status: 'Partially Matched',
   triggerStatus: '',
   trades: [{
              tradeId:31927,
              price:"0.1634',
              volume:"5',
              fee:'0.001',
              liquidityType:'Taker'
            }]
     timestamp: '2019-04-08T20:41:41.857Z',
   messageType: 'orderChange'
 }
```

### Triggered

This event is published when `Stop Limit` orders are triggered.

```
 { orderId: 7903,
   marketId: 'BTC-AUD',
   side: 'Bid',
   type: 'Limit',
   openVolume: '1.2',
   status: 'Placed',
   triggerStatus: 'Triggered',
   trades: [],
   timestamp: '2019-04-08T20:41:41.857Z',
   messageType: 'orderChange'
 }
```

**Notes:**

- `Fully Matched` and 'Partially Matched' events also include a list of trades that are the result of trade execution for that specific instance.
- In case if two or more events are published by trading engine at the same time then only the last event is published. For instance in the case of a `Stop` order being triggered and matched at the same time then a single event is published.

## Fund transfer events

Those events are published when deposit or withdraws of funds are requested by a user or approved by the system (and result in balance updates). Channel name used is `fundChange`.

```
{
  fundtransferId: 276811,
  type: 'Deposit',
  status: 'Complete',
  timestamp: '2019-04-16T01:38:02.931Z',
  amount: '0.001',
  currency: 'BTC',
  fee: '0',
  messageType: 'fundChange'
}
```

Note: status of a withdraw request is `Pending Authorization` when it's requested in the first place and before it becomes `Complete`.