General

These docs describe the v3 API for the dYdX decentralized perpetual contracts exchange. The exchange runs on an L2 (layer-2) blockchain system, and operates independently of previous dYdX protocols and systems, including the v1 and v2 APIs.

Like the previous iteration of dYdX perpetuals, the exchange uses a centralized order book, but remains non-custodial, and settles trades and liquidations in a trustless manner.

ℹ These docs describe the dYdX layer-2 perpetuals exchange. Visit the legacy docs to find API docs for the layer-1 margin trading exchange.

Layer 2: ZK-Rollups

Trades are settled in an L2 (layer-2) system, which publishes ZK (zero-knowledge) proofs periodically to an Ethereum smart contract in order to prove that state transitions within L2 are valid. Funds must be deposited to the Ethereum smart contract before they can be used to trade on dYdX.

By settling trades on L2, the exchange is able to offer much higher trade throughput and lower minimum order sizes, compared with systems settling trades directly on Ethereum (i.e. L1). This is achieved while maintaining decentralization, and the exchange is fully non-custodial.

The L2 system was developed with, and is operated jointly with, Starkware. More information about the L2 design can be found in Starkware's documentation. (Note: Some of the details described there may be specific to Starkware's previous StarkEx system and may not apply to the dYdX system.)

Data Centers

Our data centers are located in the Amazon us-east-1 region (Northern Virginia).

Number Formats

All amounts and prices in the clients and API are represented in "human readable," natural units. For example, an amount of 1.25 ETH is represented as `1.25`, and a price of $31,000.50 per BTC is represented as `31000.5`.

This differs from the dYdX v1 and v2 APIs which represented amounts and prices in terms of base token units.

Base URLs

Base URLs for API endpoints are as follows:

- **Production**: `https://api.dydx.exchange`
- **Staging (Ropsten)**: `https://api.stage.dydx.exchange`

Perpetual Contracts

The dYdX Perpetual is a non-custodial, decentralized margin product that offers synthetic exposure to a variety of assets.

Margin

Collateral is held as USDC, and the quote asset for all perpetual markets is USDC. Cross-margining is used by default, meaning an account can open multiple positions that share the same collateral. Isolated margin can be achieved by creating separate accounts (sub-accounts) under the same user.

Each market has two risk parameters, the initial margin fraction and the maintenance margin fraction, which determine the max leverage available within that market. These are used to calculate the value that must be held by an account in order to open or increase positions (in the case of initial margin) or avoid liquidation (in the case of maintenance margin).

PORTFOLIO MARGINING

There is no distinction between realized and unrealized PnL for the purposes of margin calculations. Gains from one position will offset losses from another position within the same account, regardless of whether the profitable position is closed.

MARGIN CALCULATION

The margin requirement for a single position is calculated as follows:

```
Initial Margin Requirement = abs(S × P × I)
Maintenance Margin Requirement = abs(S × P × M)
```

Where:

- `S` is the size of the position (positive if long, negative if short)
- `P` is the oracle price for the market
- `I` is the initial margin fraction for the market
- `M` is the maintenance margin fraction for the market

The margin requirement for the account as a whole is the sum of the margin requirement over each market `i` in which the account holds a position:

```
Total Initial Margin Requirement = Σ abs(S_i × P_i × I_i)
Total Maintenance Margin Requirement = Σ abs(S_i × P_i × M_i)
```

The total margin requirement is compared against the total value of the account, which incorporates the quote asset (USDC) balance of the account as well as the value of the positions held by the account:

```
Total Account Value = Q + Σ (S_i × P_i)
```

The Total Account Value is also referred to as equity.

Where:

- `Q` is the account's USDC balance (note that `Q` may be negative)
- `S` and `P` are as defined above (note that `S` may be negative)

An account cannot open new positions or increase the size of existing positions if it would lead the total account value of the account to drop below the total initial margin requirement. If the total account value ever falls below the total maintenance margin requirement, the account may be liquidated.

Free collateral is calculated as:

```
Free collateral = Total Account Value - Total Initial Margin Requirement
```

Equity and free collateral can be tracked over time using the latest oracle price (obtained from the markets websocket).

Liquidations

Accounts whose total value falls below the maintenance margin requirement (described above) may have their positions automatically closed by the liquidation engine. Positions are closed at the close price described below. Profits or losses from liquidations are taken on by the insurance fund.

CLOSE PRICE FOR LIQUIDATIONS

The close price for a position being liquidated is calculated as follows, depending whether it is a short or long position:

```
Close Price (Short) = P × (1 + (M × V / W))
Close Price (Long) = P × (1 − (M × V / W))
```

Where:

- `P` is the oracle price for the market
- `M` is the maintenance margin fraction for the market
- `V` is the total account value, as defined above
- `W` is the total maintentance margin requirement, as defined above

This formula is chosen such that the ratio `V / W` is unchanged as individual positions are liquidated.

Funding

Funding payments are exchanged between long and short traders to encourage the price of a perpetual contract to trade close to the price of the underlying. If the perpetual trades at a premium relative to the index, long traders will typically make payments to short traders, whereas if the perpetual trades at a discount relative to the index, short traders will typically make payments to long traders.

The payments are credited or debited at the start of each hour, and are included in the realized PnL for the position.

Funding payments can be found by calling Get /v3/funding and the predicted funding rate can be found by calling Get v3/markets.

FUNDING RATE UNITS

Since funding payments are exchanged every hour, the dYdX funding rate is usually represented as a 1-hour rate, which represents the return a position may expect to earn or pay every hour.

When calculating the funding rate, the premium is scaled to have a realization period of 8 hours. That means, for example, that if a certain perpetual market trades consistently at a 0.1% premium relative to the underlying, long traders may expect to pay ~0.1% every 8 hours, and short traders may expect to earn a ~0.1% return every 8 hours (not accounting for the interest rate component).

FUNDING PAYMENT CALCULATION

At the start of each hour, an account receives USDC (if `F` is positive) or pays USDC (if `F` is negative) in an amount equal to:

```
F = (-1) × S × P × R
```

Where:

- `S` is the size of the position (positive if long, negative if short)
- `P` is the oracle price for the market
- `R` is the funding rate (as a 1-hour rate)

FUNDING RATE CALCULATION

The main component of the funding rate is a premium that takes into account market activity for the perpetual. It is calculated for each market, every minute (at a random point within the minute) using the formula:

```
Premium = (Max(0, Impact Bid Price - Index Price) - Max(0, Index Price - Impact Ask Price)) / Index Price
```

Where the impact bid and impact ask prices are defined as:

```
Impact Bid Price = Average execution price for a market sell of the impact notional value
Impact Ask Price = Average execution price for a market buy of the impact notional value
```

And the impact notional amount for a market is:

```
Impact Notional Amount = 500 USDC / Initial Margin Fraction
```

For example, for a market with a 10% initial margin fraction, the impact notional value is 5,000 USDC.

At the end of each hour, the premium component is calculated as the simple average (i.e. TWAP) of the 60 premiums calculated over the course of the last hour. In addition to the premium component, each market has a fixed interest rate component that aims to account for the difference in interest rates of the base and quote currencies. The funding rate is then:

```
Funding Rate = (Premium Component / 8) + Interest Rate Component
```

Currently, the interest rate component for all dYdX markets is `0.00125%` (equivalent to `0.01%` per 8 hours).

Contract Loss Mechanisms

During periods of high volatility in the markets underlying the perpetual contracts, the value of some accounts may drop below zero before they can be liquidated.

The insurance fund is the first backstop to maintain the solvency of the system when an account has a negative balance. The account will be liquidated, and the insurance fund will take on the loss.

In the event that the insurance fund is depleted, positions with the most profit and leverage may be used to offset negative-balance accounts, in order to maintain the stability of the system.

Clients

Python and TypeScript clients are available, allowing programmatic usage of dYdX.

Python Client

INSTALLATION

Install `dydx-v3-python` from PyPI using `pip`:

```
pip install dydx-v3-python
```

USAGE

See dydxprotocol/dydx-v3-python.

See the examples folder for simple python examples.

TypeScript Client

INSTALLATION

Install `@dydxprotocol/v3-client` from NPM:

```
npm i -s @dydxprotocol/v3-client
```

USAGE

See dydxprotocol/v3-client.

See the examples folder for simple typescript examples.

Client Initialization

Initialize

```
client = Client(
    host='https://api.dydx.exchange',
    web3=Web3('...'),
    stark_private_key='01234abcd...',
)
```

The client is organized into modules, based on the type of authentication needed for different requests. The configuration options passed into the client determine which modules are available. See Authentication for more information.

ℹ Multiple methods of authorization are available, so users never need to provide private keys directly to the client, if so desired. Ethereum signatures are needed only for onboarding and managing API keys, not trading, and may be provided via a web3 provider. STARK key signatures are required for trading, and the STARK key can be held either in the client or elsewhere.

Module	Description
public	Public API endpoints. Does not require authentication.
onboarding	Endpoint to create a new user, authenticated via Ethereum key.
api_keys	Endpoints for managing API keys, authenticated via Ethereum key.
private	All other private endpoints, authenticated via API key.
eth	Calling and querying L1 Ethereum smart contracts.

The following configuration options are available:

Parameter	Description
host	The HTTP API host.
api_timeout	Timeout for HTTP requests, in milliseconds.
default_ethereum_address	(Optional) The default account for Ethereum key auth and sending Ethereum transactions.
eth_private_key	(Optional) May be used for Ethereum key auth.
eth_send_options	(Optional) Options for Ethereum transactions, see `sendTransaction`.
network_id	(Optional) Chain ID for Ethereum key auth and smart contract addresses. Defaults to `web3.net.version` if available, or `1` (mainnet).
stark_private_key	(Optional) STARK private key, used to sign orders and withdrawals.
web3	(Optional) Web3 object used for Ethereum key auth and/or smart contract interactions.
web3_account	(Optional) May be used for Ethereum key auth.
web3_provider	(Optional) Web3 provider object, same usage as `web3`.
api_key_credentials	(Optional) Dictionary containing the key, secret and passphrase required for the private module to sign requests.
crypto_c_exports_path	(Optional) For python only, will use faster C++ code to run hashing, signing and verifying. It's expected to be compiled from the `crypto_c_exports` target from Starkware's repository. See section on this below for more information.

C++ METHODS FOR FASTER STARK SIGNING

> This optimization is only available for the Python client currently.

The C++ wrapper methods in the client expect an absolute path to a Shared Object. This has to be compiled from Starkware's crypto C++ library.

Private HTTP API

Authentication

There are three levels of authentication to be considered when using dYdX. All signing can be handled directly by the client libraries.

ETHEREUM KEY AUTHENTICATION

The highest level of authentication is via an account's Ethereum private key. The Ethereum key remains in control of an account's funds while they are within the L2 system. This includes the ability to forcibly close an account's positions and exit the system, in the event that the L2 operators (dYdX and Starkware) were to unexpectedly go offline or otherwise censor requests.

Ethereum key authentication is required for the following operations:

- Register a new user or STARK key
- Create or revoke API keys
- Request a forced withdrawal or forced trade

STARK KEY AUTHENTICATION

Within the L2 system, authentication is handled by a separate key pair, known as the account's STARK key pair.

STARK key authentication is required for the following operations:

- Place an order
- Withdraw funds

API KEY AUTHENTICATION

The third level of authentication consists of the API key, secret and passphrase which are used solely to authenticate API requests made to dYdX. This includes operations such as canceling orders or retrieving an account's fills, which do not affect the L2 system.

When a user onboards via `POST v3/onboarding`, the server will use the signature as a seed to determinstically generate default API key credentials. An API key includes three fields:

- `key`: UUID identifying the credentials.
- `secret`: Secret string used to generate HMACs, not sent with requests.
- `passphrase`: Secret string sent with each request, used to encrypt/decrypt the secret in our DB, and never stored in our DB.

API keys can be added and managed via the `/v3/api-keys` endpoints.

All requests which are not signed by an Ethereum key and which are made to private endpoints require an API key signature.

STARK KEY CRYPTOGRAPHY

The STARK and API keys are ECDSA key pairs on the STARK curve. More info on the cryptography used on L2 is available in Starkware's documentation.

Creating and Signing Requests

Note that the Python and TypeScript clients can generate all required signatures.

Within the private HTTP API, there are three groups of endpoints which each require different headers and authentication.

(Separately, and in addition to the above, STARK signatures are required for orders and withdrawals. For details, please refer to the Python and TypeScript reference implementations.)

ONBOARDING ENDPOINT: `POST V3/ONBOARDING`

Request Headers

Header	Required?	Description
DYDX-SIGNATURE	yes	Ethereum key authentication
DYDX-ETHEREUM-ADDRESS	yes	Ethereum address of the user

Signing

The header `DYDX-SIGNATURE` is an EIP-712 Ethereum signature on a static message containing the fields:

- `action`: The string `DYDX-ONBOARDING`.
- `onlySignOn`: The string `https://trade.dydx.exchange`.

See reference implementations: [Python] [TypeScript]

ETHEREUM KEY PRIVATE ENDPOINTS

This group includes the `POST` and `DELETE` `v3/api-keys` endpoints for managing API keys. Like the onboarding endpoint, requests to these endpoints require signatures by the user's Ethereum key.

Request Headers

Header	Required?	Description
DYDX-SIGNATURE	yes	Ethereum key authentication
DYDX-ETHEREUM-ADDRESS	yes	Ethereum address of the user
DYDX-TIMESTAMP	yes	ISO timestamp of when the request was signed. Must be within 30 seconds of the server time.

Signing

The header `DYDX-SIGNATURE` is an EIP-712-compliant Ethereum signature on a message containing the fields:

- `method`: The name of the HTTP method used, uppercase (e.g. `GET`).
- `requestPath`: The API endpoint path, beginning with `/v3/`.
- `body`: The HTTP request body (normally empty for `GET` and `DELETE`).
- `timestamp`: Equal to the header `DYDX-TIMESTAMP`.

See reference implementations: [Python] [TypeScript]

API KEY PRIVATE ENDPOINTS

All private endpoints not listed above fall in this category, and must be authenticated via an API key.

Request Headers

Header	Required?	Description
DYDX-SIGNATURE	yes	HMAC of the request.

Header	Required?	Description
DYDX-API-KEY	yes	Api key for the account.
DYDX-TIMESTAMP	yes	ISO timestamp of when the request was signed. Must be within 30 seconds of the server time.
DYDX-PASSPHRASE	yes	The `passphrase` field of the API key.
DYDX-ACCOUNT-NUMBER	no	Account number used to scope the request. Defaults to zero.

Signing

The `DYDX-SIGNATURE` is a SHA-256 HMAC produced as described below, and encoded as a `Base64` string.

A SHA-256 HMAC is created using the API key `secret` and the message `timestamp + method + requestPath + body` defined as follows:

- `timestamp`: The `DYDX-TIMESTAMP` header, which must be within 30 seconds of the server time.
- `method`: The name of the HTTP method used, uppercase (e.g. `GET`).
- `requestPath`: The API endpoint path, beginning with `/v3/`.
- `body`: The HTTP request body (normally empty for `GET` and `DELETE`).

The HMAC should be encoded as a Base64 string and sent as the `DYDX-SIGNATURE` header.

See reference implementations: [Python] [TypeScript]

Rate Limits

Request	Limit
`GET v3/*`	100 requests per 10 seconds.
`POST v3/orders`	100 requests per 10 seconds per market.
`DELETE v3/orders`	3 requests per 10 seconds per market.
`DELETE v3/orders/:id`	250 requests per 10 seconds per market.
`PUT v3/emails/send-verification-email`	2 requests for 10 minutes.
`All other requests`	10 requests per minute.

These rate limits are subject to change.

Additionally, if more than 100 requests are made in 10 seconds, requests will be blocked by a global rate limiter for one minute. Please make use of the WebSockets API if you need real-time data.

Onboarding

OVERVIEW

A few steps are required of all accounts before they can begin trading:

1. Create a user, providing a STARK public key to be associated with the main account.

2. Request registration signature from dYdX.
3. Send registration request to the L1 smart contract.
4. Approve collateral token allowance on the L1 smart contract.
5. Deposit collateral token to the L1 smart contract.

All of these steps are supported by the Python and TypeScript clients. See the Python integration tests for an example of onboarding and usage of various endpoints.

Create User

```
onboarding_information = client.onboarding.create_user(
  # Optional if stark_private_key was provided.
  stark_public_key='012340bcd...',
  stark_public_key_y_coordinate='01234abcd...',
  # Optional if eth_private_key or web3.eth.defaultAccount was provided.
  ethereum_address='ethereumAddress',
)
```

HTTP REQUEST

`POST v3/onboarding`

Programmatic users of the API must take care to store private STARK and API keys securely. dYdX does not store any private keys. Using the default key generation methods (such as `derive_stark_key`) ensures keys can be easily recovered by Ethereum key holder. If you generate your STARK key through other means, you must be careful not to lose it, or your funds may be inaccessible for a period of time.

Description: Onboard a user so they can begin using dYdX V3 API. This will generate a user, account and derive a key, passphrase and secret from the signature.

REQUEST

Parameter	Description
starkKey	Public starkKey associated with the key-pair you created.
starkKeyYCoordinate	Public starkKey Y-Coordinate associated with the key-pair you created.
ethereumAddress	Ethereum address associated with the user being created.
referredByAffiliateLink	(Optional) Link to affiliate the user was referred by.

RESPONSE

Parameter	Description
apiKey	See ApiKeyCredentials.
user	See User.
account	See Account.

Derive StarkKey

Derive StarkKey

```
key_pair_with_y_coordinate = client.onboarding.derive_stark_key(
  # Optional if eth_private_key or web3.eth.defaultAccount was provided.
  ethereum_address='ethereumAddress',
)
```

This method does not access the dYdX API. This is used by the frontend app to derive the STARK key pair in a way that is recoverable. Programmatic traders may optionally derive their STARK key pair in the same way.

REQUEST

Parameter	Description
ethereumAddress	Ethereum address associated with the user being created.

RESPONSE

Parameter	Description
keyPairWithYCoordinate	KeyPairWithYCoordinate.

KEYPAIRWITHYCOORDINATE

field	type	description
publicKey	string	The x-coordinate of the publicKey.
publicKeyYCoordinate	string	The y-coordinate of the publicKey.
privateKey	string	The privateKey for the key pair.

Recover Default API Credentials

Recover Default API Credentials

```
api_credentials = client.onboarding.recover_default_api_key_credentials(
  # Optional if eth_private_key or web3.eth.defaultAccount was provided.
  ethereum_address='ethereumAddress',
)
```

This method does not access the dYdX API. This can be used to recover the default API key credentials, which are the same set of credentials used in the dYdX frontend.

REQUEST

Parameter	Description
ethereumAddress	Ethereum address associated with the user being created.

RESPONSE

Parameter	Description

Parameter	Description
apiCredentials	ApiKeyCredentials.

APIKEYCREDENTIALS

field	type	description
key	string	UUID identifying the credentials.
secret	string	Secret string used to generate HMACs.
passphrase	string	Secret string sent with each request.

Get Registration

Get Registration

```
signature = client.private.get_registration()
```

HTTP REQUEST

`GET v3/registration`

Description: Gets the dYdX provided Ethereum signature required to send a registration transaction to the Starkware smart contract.

RESPONSE

Parameter	Description
signature	Ethereum signature authorizing the user's Ethereum address to register for the corresponding position id.

Register API Key

Register API Key

```
api_key_response = client.api_keys.create_api_key(
  # Optional if eth_private_key or web3.eth.defaultAccount was provided.
  ethereum_address='0x0123...',
)
```

HTTP REQUEST

`POST v3/api-keys`

Description: Create new API key credentials for a user.

RESPONSE

Parameter	Description
apiKey	ApiKeyCredentials.

Get API Keys

Get API Keys

```
api_keys = client.private.get_api_keys()
```

HTTP REQUEST

`GET v3/api-keys`

Description: Get all api keys associated with an Ethereum address.

> Note that this endpoint is in the private module, unlike the methods to create or revoke API keys.

RESPONSE

Parameter	Description
apiKeys	Array of apiKey strings corresponding to the ethereumAddress in the request.

Delete API Key

Delete API Key

```
client.api_keys.delete_api_key(
  api_public_key='290decd9-548b-...',
  # Optional if eth_private_key or web3.eth.defaultAccount was provided.
  ethereum_address='0x0123...',
)
```

HTTP REQUEST

`DELETE v3/api-keys`

Description: Delete an api key by key and Ethereum address.

REQUEST

Parameter	Description
apiKey	Public api key being deleted.
ethereumAdddress	Ethereum address the api key is associated with.

RESPONSE

Returns a 200 on success.

Get User

Get User

```
user = client.private.get_user()
```

HTTP REQUEST

`GET v3/users`

Description: return the user and user information.

RESPONSE

Parameter	Description
ethereumAddress	The 20-byte Ethereum address.
isRegistered	True if the user is registered on the starkware smart contract. This is false otherwise.
email	Email address.
username	User defined username.
referredByAffiliateLink	The affiliate link that referred this user, or null if the user was not referred.
makerFeeRate	The fee rate the user would be willing to take as the maker. Note, 1% would be represented as 0.01.
takerFeeRate	The fee rate the user would be willing to take as the taker. Note, 1% would be represented as 0.01.
makerVolume30D	The user's thirty day maker volume. Note, this is in USD (eg $12.34 -> 12.34).
takerVolume30D	The user's thirty day maker volume. Note, this is in USD (eg $12.34 -> 12.34).
fees30D	The user's thirty day fees. Note, this is in USD (eg $12.34 -> 12.34).
userData	The user's unstructured user data.
dydxTokenBalance	The user's DYDX token holdings.
stakedDydxTokenBalance	The user's staked DYDX token holdings

Update User

Update user

```
user = client.private.update_user(
  user_data={},
  email='user@example.com',
  username='username',
)
```

HTTP REQUEST

`PUT v3/users`

Description: Update user information and return the updated user.

Parameter	Description
userData	User metadata in a JSON blob.

Parameter	Description
email	(Optional) Email to be used with the user.
username	(Optional) Username to be used for the user.
isSharingUsername	(Optional) Share username publically on leaderboard rankings.
isSharingAddress	(Optional) Share ETH address publically on leaderboard rankings.

RESPONSE

Parameter	Description
user	See User.

Create An Account

Create Account

```
client.private.create_account(
  stark_public_key='701234abcd...',
  stark_public_key_y_coordinate='1234abcd...',
)
```

An account will be created automatically during onboarding so this call is not necessary to get started.

HTTP REQUEST

`POST v3/accounts`

Description: Create an account with a given starkKey.

REQUEST

Parameter	Description
starkKey	Public starkKey associated with the key-pair you created.
starkKeyYCoordinate	Public starkKey Y-Coordinate associated with the key-pair you created.

RESPONSE

Parameter	Description
account	See Account.

Get Account

Get Account

```
account = client.private.get_account(
```

```
  # Optional if eth_private_key or web3.eth.defaultAccount was provided.
  ethereum_address='0x0123...',
)
```

HTTP REQUEST

`GET v3/accounts/:id`

Description: Get an account for a user by id. Using the client, the id will be generated with client information and an Ethereum address.

REQUEST

Parameter	Description
ethereumAddress	Ethereum address associated with an account.

RESPONSE

Parameter	Description
starkKey	Public StarkKey associated with an account.
positionId	Starkware-specific positionId.
equity	The amount of equity (value) in the account. Uses balances and index prices to calculate.
freeCollateral	The amount of collateral that is withdrawable from the account.
quoteBalance	Human readable quote token balance. Can be negative.
pendingDeposits	The sum amount of all pending deposits.
pendingWithdrawals	The sum amount of all pending withdrawal requests.
openPositions	See Positions. Note, markets where the user has no position are not returned in the map.
accountNumber	Unique accountNumber for the account.
id	Unique id of the account hashed from the userId and the accountNumber.

Get Account Leaderboard PNLs

Get Account Leaderboard PNLs

HTTP REQUEST

`GET v3/accounts/leaderboard-pnl/:period`

Description: Get an account's personal leaderboard pnls.

REQUEST

Parameter	Description
period	Which time period of PNLs. "DAILY" or "WEEKLY" or "MONTHLY" or "ALLTIME"

RESPONSE

Parameter	Description

Parameter	Description
absolutePnl	The account's latest updated absolute PNL
percentPnl	The account's latest updated percent PNL
absoluteRank	User's absolute PNL rank. `null` if not ranked.
percentRank	User's percent PNL rank. `null` if not ranked.
updatedAt	When these leaderboard PNLs were last updated
accountId	The account the PNLs are for

Get Accounts

Get Account

```
accounts = client.private.get_accounts()
```

HTTP REQUEST

`GET v3/accounts`

Description: Get all accounts for a user.

RESPONSE

Parameter	Description
accounts	See Account. Returns an array of Accounts.

Get Positions

Get Positions

```
from dydx3.constants import MARKET_BTC_USD
from dydx3.constants import POSITION_STATUS_OPEN

all_positions = client.private.get_positions(
  market=MARKET_BTC_USD,
  status=POSITION_STATUS_OPEN,
)
```

HTTP REQUEST

`GET v3/positions`

Description: Get all current positions for a user by specified query parameters.

REQUEST

Parameter	Description

Parameter	Description
market	(Optional) Market of the position.
status	(Optional) Status of the position. Can be `OPEN`, `CLOSED` or `LIQUIDATED`.
limit	(Optional) The maximum number of positions that can be fetched via this request. Note, this cannot be greater than 100.
createdBeforeOrAt	(Optional) Set a date by which the positions had to be created.

RESPONSE

Parameter	Description
market	The market of the position.
status	The status of the position.
side	The side of the position. `LONG` or `SHORT`.
size	The current size of the position. Positive if long, negative if short, 0 if closed.
maxSize	The maximum (absolute value) size of the position. Positive if long, negative if short.
entryPrice	Average price paid to enter the position.
exitPrice	Average price paid to exit the position.
unrealizedPnl	The unrealized pnl of the position in quote currency.
realizedPnl	The realized pnl of the position in quote currency.
createdAt	Timestamp of when the position was opened.
closedAt	Timestamp of when the position was closed.
netFunding	Sum of all funding payments for this position.
sumOpen	Sum of all trades sizes that increased the size of this position.
sumClose	Sum of all trades sizes that decreased the size of this position.

Get Transfers

Get Transfers

```
from dydx3.constants import ACCOUNT_ACTION_DEPOSIT

transfers = client.private.get_transfers(
  transfer_type=ACCOUNT_ACTION_DEPOSIT,
  limit=50,
)
```

HTTP REQUEST

`GET v3/transfers`

Description: Get transfers for a user, limited by query parameters.

REQUEST

Parameter	Description
transferType	(Optional) Type of the transfer. Can be `DEPOSIT`, `WITHDRAWAL` or `FAST_WITHDRAWAL`.
limit	(Optional) The maximum number of transfers that can be fetched via this request. Note, this cannot be greater than 100.
createdBeforeOrAt	Latest that the transfers could have been created.

RESPONSE

Parameter	Description
id	Unique id assigned by dYdX.
type	Type of the transfer. Will be `DEPOSIT`, `WITHDRAWAL` or `FAST_WITHDRAWAL`.
debitAsset	Asset that was debited (USDC, USDT, USD, etc).
creditAsset	Asset that was credited (USDC, USDT, USD, etc).
debitAmount	Amount that was sent in for the deposit in debitAsset.
creditAmount	Amount that was credited to the account in creditAsset.
transactionHash	Ethereum transaction hash of the transfer.
status	Status of the transfer. Will be `PENDING` or `CONFIRMED`.
createdAt	Timestamp when created.
confirmedAt	Timestamp when confirmed.
clientId	ClientId of transfer.
fromAddress	The Ethereum address the transfer is from.
toAddress	The Ethereum address the transfer is for.

Fast vs. Slow Withdrawal

The normal process for withdrawing from L2 to L1 requires waiting for a block of L2 transactions to be collected, and the zero-knowledge proof for the bock to be constructed and verified on-chain.

Using the fast withdrawal process, users can get their funds on L1 much faster by essentially trading their L2 funds to an “LP” account operated by dYdX, in order to receive immediate liquidity on L1. Since the LP must then recycle these funds from L2 to L1 via the regular withdrawal process, dYdX is only able to process a certain volume of fast withdrawals within a given period of time.

Create Withdrawal

Create Withdrawal

```
from dydx3.constants import ASSET_USDC
```

```
withdrawal = client.private.create_withdrawal(
  position_id=1,
  amount='100',
  asset=ASSET_USDC,
  expiration_epoch_seconds=1613988637,
)
```

HTTP REQUEST

`POST v3/withdrawals`

Description: Create a withdrawal from an account.

> If not withdrawing the entirety of your balance, there is a minimum withdrawal amount. Currently that amount is 100 USDC.

REQUEST

Parameter	Description
positionId	The unique id of the position the withdrawal is associated with.
amount	Amount to be withdrawn.
asset	Asset being withdrawn. Can currently only be `USDC`.
expiration	Datetime at which the withdrawal expires if it has not been completed.
clientId	(Optional) Unique id of the client associated with the withdrawal. Must be <= 40 characters.
signature	(Optional) The starkware signature for the withdrawal. If not included, will be done by the client.

RESPONSE

Parameter	Description
withdrawal	See Transfers.

Create Fast Withdrawal

Create Fast Withdrawal

```
from dydx3.constants import ASSET_USDC

fast_withdrawal = client.private.create_fast_withdrawal(
  credit_asset=ASSET_USDC,
  credit_amount='100',
  debit_amount='110',
  to_address='0x98ab...',
  lp_position_id='2',
  expiration_epoch_seconds=1613988637,
  signature='0abc12...',  # Optional if stark_private_key was provided to client.
)
```

HTTP REQUEST

`POST v3/fast-withdrawals`

Description: Create a fast-wthdrawal.

REQUEST

Parameter	Description
creditAsset	Asset being withdrawn. Can currently only be `USDC`.
creditAmount	Amount that is expected.
debitAmount	Amount offered in `USDC` for the credit amount.
toAddress	Address to be credited
lpPositionId	LP Position Id of the debit account.
expiration	Datetime at which the withdrawal expires if it has not been completed.
signature	Starkware signature for the fast withdrawal. Currently can be anything.
clientId	(Optional) Unique id of the client associated with the withdrawal. Must be <= 40 characters.

RESPONSE

Parameter	Description
withdrawal	See Transfers.

Order Types

Type	Description
MARKET	Market order (must be FOK or IOC).
LIMIT	Limit order.
STOP	Stop limit order.
TRAILING_STOP	Trailing stop limit order.
TAKE_PROFIT	Trailing stop limit order.
LIQUIDATED	Indicates the account was liquidated (fills only).
LIQUIDATION	Indicates the account took over a liquidated account (fills only).

Create A New Order

Create Order

```
from dydx3.constants import MARKET_BTC_USD
from dydx3.constants import ORDER_SIDE_SELL
from dydx3.constants import ORDER_TYPE_LIMIT
from dydx3.constants import TIME_IN_FORCE_GTT
```

```
placed_order = client.private.create_order(
  position_id=1,
  market=MARKET_BTC_USD,
  side=ORDER_SIDE_SELL,
  order_type=ORDER_TYPE_LIMIT,
  post_only=False,
  size='100',
  price='18000',
  limit_fee='0.015',
  expiration_epoch_seconds=1613988637,
  time_in_force=TIME_IN_FORCE_GTT,
)
```

HTTP REQUEST

`POST v3/orders`

Description: Create a new order.

REQUEST

Parameter	Description
positionId	StarkWare-specific positionId. This is required for signing and can be fetched from the account endpoints.
market	Market of the order.
side	Either `BUY` or `SELL`.
type	The type of order. This can be `MARKET`, `LIMIT`, `STOP_LIMIT`, `TRAILING_STOP` or `TAKE_PROFIT`.
postOnly	Whether the order should be canceled if it would fill immediately on reaching the matching-engine.
size	Size of the order, in base currency (i.e. an ETH-USD position of size 1 represents 1 ETH).
price	Worst accepted price of the base asset in USD.
limitFee	Is the highest accepted fee for the trade. See below for more information.
expiration	Time at which the order will expire if not filled. This is the Good-Til-Time and is accurate to a granularity of about 15 seconds.
timeInForce	(Optional) One of `GTT` (Good til time), `FOK`(Fill or kill) or `IOC` (Immediate or cancel). This will default to `GTT`.
cancelId	(Optional) The id of the order that is being replaced by this one.
triggerPrice	(Optional) The triggerPrice at which this order will go to the matching-engine.
trailingPercent	(Optional) The percent that the triggerPrice trails the indexPrice of the market.
clientId	(Optional) Unique id assigned by the client. Must be <= 40 characters.
signature	(Optional) Signature for the order, signed with the account's STARK private key.

ℹ Specifying `cancelId` will cause the order matching `cancelId` to be canceled **atomically** (and immediately before) with the new order being placed. This can be used to always have available liquidity on the book when market making. The new order will still be placed even if the old order was already filled.

❗ All Stop-Limit, Take-Profit and Trailing-Stop orders must have timeInForce `FOK`.

RESPONSE

Parameter	Description
order	See order.

Order LimitFee

The `limitFee` is the highest fee a user would be willing to accept on an order. This should be in decimal form (i.e. 0.1 is 10%). To see current fees, call GET /v3/users and the maker/taker fee rates show what fees will be. If the order is `postOnly` dYdX will validate against `makerFeeRate` only. The opposite is true if the order is `FOK` or `IOC` - dYdX will only validate against `takerFeeRate`. Otherwise, dYdX assesses against the maximum of maker and taker fee rate.

Tick Size and Minimum Size

TICK SIZE

Each market has a specified `tickSize`. Order `price` must be a multiple of the tickSize. THe same applies to `triggerPrice` and `trailingPercent` if either of these are not null.

MINIMUM SIZE

Each market has a specified `minOrderSize`. Order `size` must be not be less than the minOrderSize.

Order Deletion

Canceled orders older than one month are deleted from the dYdX database.

Cancel An Order

Cancel an order

```
client.private.cancel_order(order_id='0x0000')
```

HTTP REQUEST

`DELETE v3/orders/:id`

Description: Cancel an order by its unique id.

REQUEST

Parameter	Description
orderId	Unique id of the order to be canceled.

RESPONSE

The endpoint returns with status code `200` once the order has been queued for cancelation. The order's status will be updated after the cancelation has been processed by the matching engine.

Parameter	Description
cancelOrder	See order.

Cancel Orders

Cancel orders

```
from dydx3.constants import MARKET_BTC_USD

client.private.cancel_all_orders(market=MARKET_BTC_USD)
```

HTTP REQUEST

`DELETE v3/orders`

Description: Either bulk cancel all orders or just all orders for a specific market.

REQUEST

Parameter	Description
market	(Optional) Market of the orders being canceled.

RESPONSE

The endpoint returns with status code `200` once the orders have been queued for cancelation. The orders' statuses will be updated after the cancelations have been processed by the matching engine.

Parameter	Description
cancelOrders	Returns an array of orders to be canceled. See order.

Get Orders

Get Orders

```
from dydx3.constants import MARKET_BTC_USD
from dydx3.constants import ORDER_SIDE_SELL
from dydx3.constants import ORDER_TYPE_LIMIT
from dydx3.constants import ORDER_STATUS_OPEN

all_orders = client.private.get_orders(
  market=MARKET_BTC_USD,
  status=ORDER_STATUS_OPEN,
  side=ORDER_SIDE_SELL,
  type=ORDER_TYPE_LIMIT,
```

```
    limit=50,
)
```

HTTP REQUEST

`GET v3/orders`

Description: Get active (not filled or canceled) orders for a user by specified parameters.

REQUEST

Parameter	Description
market	(Optional) Market of the order.
status	(Optional) A list of statuses to filter by. Must be in the subset `PENDING`
side	(Optional) Either `BUY` or `SELL`.
type	(Optional) The expected type of the order. This can be `LIMIT`, `STOP`, `TRAILING_STOP` or `TAKE_PROFIT`.
limit	(Optional) The maximum number of orders that can be fetched via this request. Note, this cannot be greater than 100.
createdBeforeOrAt	(Optional) Set a date by which the orders had to be created.

RESPONSE

Parameter	Description
orders	An array of orders. See order below.

ORDER

Parameter	Description
id	The unique id assigned by dYdX.
clientId	The unique id assigned by the client.
accountId	The id of the account.
market	Market of the fill.
side	Either `BUY` or `SELL`.
price	The price of the order. Must adhere to the market's tick size.
triggerPrice	The trigger price of the order. Must adhere to the market's tick size.
trailingPercent	Used for trailing stops. Percent drop from maximum price that will trigger the order.
size	Total size (base currency) of the order
remainingSize	Size of order not yet filled.
type	The type of the fill.
createdAt	Timestamp when the fill was created.
unfillableAt	Time order was either filled or canceled.
expiresAt	Time order will expire.

Parameter	Description
status	See order statuses below.
timeInForce	(Optional) One of `GTT` (Good til time), `FOK`(Fill or kill) or `IOC` (Immediate or cancel). This will default to `GTT`.
postOnly	If the order will cancel if it would take the position of `TAKER`.
cancelReason	See cancel reasons below.

ORDER STATUSES

Status	Description
PENDING	Order has yet to be processed by the matching engine.
OPEN	Order is active and on the orderbook. Could be partially filled.
FILLED	Fully filled.
CANCELED	Canceled, for one of the cancel reasons. Could be partially filled.
UNTRIGGERED	Triggerable order that has not yet been triggered.

CANCEL REASONS

Reason	Description
UNDERCOLLATERALIZED	Order would have led to an undercollateralized state for the user.
EXPIRED	Order expired.
USER_CANCELED	Order was canceled by the user.
SELF_TRADE	Order would have resulted in a self trade for the user.
FAILED	An internal issue caused the order to be canceled.
COULD_NOT_FILL	A FOK or IOC order could not be fully filled.
POST_ONLY_WOULD_CROSS	A post-only order would cross the orderbook.

Get Order By Id

Get Order By Id

```
order = client.private.get_order_by_id('foo')
```

HTTP REQUEST

`GET v3/orders/:id`

Description: Get an order by `id` from the active orderbook and order history.

REQUEST

Parameter	Description

Parameter	Description
id	Unique id of the order

RESPONSE

Parameter	Description
order	See order.

Get Order by ClientId

Get Order By ClientId

```
order = client.private.get_order_by_client_id('clientId')
```

HTTP REQUEST

`GET v3/orders/client/:id`

Description: Get an order by `clientId` from the active orderbook and order history.

REQUEST

Parameter	Description
id	Unique clientId of the order

RESPONSE

Parameter	Description
order	See order.

Get Fills

Get Fills

```
from dydx3.constants import MARKET_BTC_USD

all_fills = client.private.get_fills(
  market=MARKET_BTC_USD,
)
```

HTTP REQUEST

`GET v3/fills`

Description: Get Fills for a user by specified parameters.

REQUEST

Parameter	Description

Parameter	Description
market	(Optional) Market of the fills.
orderId	(Optional) Unique order id. Will only fetch a single order.
limit	(Optional) The maximum number of fills that can be fetched via this request. Note, this cannot be greater than 100.
createdBeforeOrAt	(Optional) Set a date by which the fills had to be created.

RESPONSE

Parameter	Description
fills	Array of fills. See below for an individual example.

FILL

Parameter	Description
id	The unique id assigned by dYdX.
side	Either `BUY` or `SELL.`
liquidity	Either `MAKER` or `TAKER.`
type	The type of the fill.
market	Market of the fill.
orderId	Id of the order which caused this fill. null if type is `LIQUIDATED` or `LIQUIDATION`.
price	The price the fill occurred at (in quote / base currency).
size	Size that was filled (in base currency).
fee	Fee that was charged (in quote currency).
createdAt	Timestamp when the fill was created.

Get Funding Payments

Get Funding Payments

```
from dydx3.constants import MARKET_BTC_USD

funding_payments = client.private.get_funding_payments(
  market=MARKET_BTC_USD,
  limit=75,
)
```

HTTP REQUEST

`GET v3/funding`

Description: Get Funding Payments made to an account.

REQUEST

Parameter	Description
market	(Optional) Market of the funding payments.
limit	(Optional) The maximum number of funding payments that can be fetched via this request. Note, this cannot be greater than 100.
effectiveBeforeOrAt	(Optional) Set a date by which the funding payments had to be created.

RESPONSE

Parameter	Description
market	Market corresponding to the funding payment.
payment	Change in the `quoteBalance` of the account. Positive if the user received funding and negative if the user paid funding.
rate	Funding rate at the time of this payment (as a 1-hour rate).
positionSize	User's position size at the time of this funding payment. positive if long, negative if short.
price	Oracle price used to calculate this funding payment.
effectiveAt	Time of this funding payment.

Get Historical PNL Ticks

Get Historical PNL Ticks

```
historical_pnl = client.private.get_historical_pnl(
  created_before_or_at='2021-04-09T22:02:46+0000',
)
```

HTTP REQUEST

`GET v3/historical-pnl`

Description: Get Historical PNL for an account during an interval.

The max interval of ticks is 1 month. If a single time value is provided, the other value will default to one month away from said value (i.e. set createdBeforeOrAt and createdOnOrAfter will be a month before). If neither value is set, the interval will be the current past roughly 30 days.

REQUEST

Parameter	Description
effectiveBeforeOrAt	(Optional) Used for setting a ending bounds on the ticks.
effectiveAtOrAfter	(Optional) Used for setting a starting bounds on the ticks.

RESPONSE

Parameter	Description

Parameter	Description
historicalPnl	Array of HistoricalAggregatedPnl. See "HistoricalAggregatedPnl" below.

HISTORICALAGGREGATEDPNL

Parameter	Description
equity	The total account equity.
totalPnl	The total PNL for the account since inception.
createdAt	When the tick was recorded.
netTransfers	The value into or out of the account of transfers since the last interval.
accountId	Account the tick is for.

Get Trading Rewards

Get Trading Rewards

```
rewards = client.private.get_trading_rewards(
  epoch=0,
)
```

HTTP REQUEST

`GET v3/rewards/weight`

Description: Get the rewards weight of a given epoch.

REQUEST

Parameter	Description
epoch	(Optional) Epoch number to request rewards data for. Defaults to current epoch.

RESPONSE

Parameter	Description
epoch	ID of the epoch.
epochStart	Time when the epoch starts.
epochEnd	Time when the epoch ends.
fees	See "Fees" below.
openInterest	See "OpenInterest" below.
weight	See "Weight" below.
totalRewards	The total number of tokens that will be given out at the end of the epoch.
estimatedRewards	The user's estimated number of dYdX tokens as rewards.

WEIGHT

Parameter	Description
weight	The user's current weight score for this epoch.
totalWeight	The sum of the weight score of all users for this epoch.

FEES

Parameter	Description
feesPaid	The USD amount of fees paid by the user in the epoch.
totalFeesPaid	The total USD amount of fees paid by all users in the epoch.

OPENINTEREST

Parameter	Description
averageOpenInterest	The average open interest for the user.
totalAverageOpenInterest	The total average open interest for all users.

Get Liquidity Provider Rewards

Get Liquidity Provider Rewards

```
rewards = client.private.get_liquidity_provider_rewards(
  epoch=0,
)
```

HTTP REQUEST

`GET v3/rewards/liquidity`

Description: Get the liquidity rewards of a given epoch.

REQUEST

Parameter	Description
epoch	(Optional) Epoch number to request rewards data for. Defaults to current epoch.

RESPONSE

Parameter	Description
epoch	ID of the epoch.
epochStart	Time when the epoch starts.
epochEnd	Time when the epoch ends.
markets	Map of market name to rewards for that market. See "LiquidityRewards" below.

LIQUIDITYREWARDS

Parameter	Description
market	The market for which the rewards are for.
uptime	The ratio of uptime (non-zero scores) that the user has for this market.
score	The user's total score for this market.
totalScore	The total score of all liquidity providers who are eligible for liquidity rewards.
totalRewards	The total number of tokens that will be given out at the end of the epoch for this market.
estimatedRewards	The user's estimated number of dYdX tokens as rewards for this market.

Get Retroactive Mining Rewards

Get Retroactive Mining Rewards

```
rewards = client.private.get_retroactive_mining_rewards()
```

HTTP REQUEST

`GET v3/rewards/retroactive-mining`

Description: Get the retroactive mining rewards of a given epoch.

RESPONSE

Parameter	Description
epoch	Will always be '0'.
epochStart	Time when the epoch starts.
epochEnd	Time when the epoch ends.
retroactiveMining	See "RetroactiveMiningRewards" below.
estimatedRewards	The user's estimated number of dYdX tokens as rewards.

RETROACTIVEMININGREWARDS

Parameter	Description
allocation	The number of allocated dYdX tokens for the user.
targetVolume	The user's required trade volume (in USD) to be able to claim the allocation.
volume	The user's current total trade volume (in USD) in the epoch.

Public HTTP API

Get Markets

Get Markets

```
markets = client.public.get_markets()
```

HTTP REQUEST

`GET v3/markets`

Description: Get one or all markets as well as metadata about each retrieved market.

REQUEST

Parameter	Description
market	(Optional): Specific market to be fetched.

RESPONSE

MARKET

Parameter	Description
markets	Map of market objects. See below for individual market.

Parameter	Description
market	Symbol of the market.
status	Status of the market. Can be one of `ONLINE`, `OFFLINE`, `POST_ONLY` or `CANCEL_ONLY`.
baseAsset	Symbol of the base asset. e.g. "BTC".
quoteAsset	Symbol of the quote asset. e.g. "BTC".
stepSize	The minimum step size (in base currency) of trade sizes for the market.
tickSize	The Tick size of the market.
indexPrice	The current index price of the market.
oraclePrice	The current oracle price of the market.
priceChange24H	The absolute price change of the index price over the past 24 hours.
nextFundingRate	The predicted next funding rate (as a 1-hour rate).
nextFundingAt	The timestamp of the next funding update.
minOrderSize	Minimum order size for the market.
type	Type of the market. This will always be `PERPETUAL` for now.
initialMarginFraction	The margin fraction needed to open a position.
maintenanceMarginFraction	The margin fraction required to prevent liquidation.
baselinePositionSize	The max position size (in base token) before increasing the initial-margin-fraction.
incrementalPositionSize	The step size (in base token) for increasing the `initialMarginFraction` by (`incrementalInitialMarginFraction` per step)
incrementalInitialMarginFraction	The increase of `initialMarginFraction` for each `incrementalPositionSize` above the `baselinePositionSize` the position is.

Parameter	Description
maxPositionSize	The max position size for this market in base token.
volume24H	The USD volume of the market in the previous 24 hours.
trades24H	The number of trades in the market in the previous 24 hours.
openInterest	The open interest in base token.
assetResolution	The asset resolution is the number of quantums (Starkware units) that fit within one "human-readable" unit of the asset.

Get Orderbook

Get Trades

```
from dydx3.constants import MARKET_BTC_USD

orderbook = client.public.get_orderbook(
  market=MARKET_BTC_USD,
)
```

HTTP REQUEST

`GET v3/orderbook/:market`

Returns bids and asks which are each Orderbook order arrays (price and size).

Description: Returns the active orderbook for a market. All bids and asks that are fillable are returned.

REQUEST

Parameter	Description
market	Market of the Orderbook.

RESPONSE

Parameter	Description
bids	See Orderbook Order below. Sorted by price in descending order.
asks	See Orderbook Order below. Sorted by price in ascending order.

ORDERBOOK ORDER

Parameter	Description
price	The price of the order (in quote / base currency).
size	The size of the order (in base currency).

Get Trades

Get Trades

```
from dydx3.constants import MARKET_BTC_USD

all_trades = client.public.get_trades(
  market=MARKET_BTC_USD,
)
```

HTTP REQUEST

`GET v3/trades/:market`

Description: Get Trades by specified parameters. Passing in all query parameters to the HTTP endpoint would look like: `GET v3/trades/BTC-USD?startingBeforeOrAt=2021-09-05T17:33:43.163Z&limit=1`.

ℹ Trades will include information for all users and as such includes less information on individual transactions than the fills endpoint.

REQUEST

Parameter	Description
market	Market of the trades.
startingBeforeOrAt	(Optional): Set a date by which the trades had to be created.
limit	(Optional): The number of candles to fetch (Max 100).

RESPONSE

Parameter	Description
trades	An array of trades. See trade below

TRADE

Parameter	Description
side	Either `BUY` or `SELL`.
size	The size of the trade.
price	The price of the trade.
createdAt	The time of the trade.

Get Fast Withdrawal Liquidity

Get Fast Withdrawal

```
fast_withdrawals_info = client.public.get_fast_withdrawal()
```

HTTP REQUEST

`GET v3/fast-withdrawals`

Description: Returns a map of all LP provider accounts that have available funds for fast withdrawals. Given a `debitAmount` and asset the user wants sent to L1, this endpoint also returns the predicted amount of the desired asset the user will be credited on L1. Given a `creditAmount` and asset the user wants sent to L1, this endpoint also returns the predicted amount the user will be debited on L2.

REQUEST

Parameter	Description
creditAsset	(Optional): The asset that would be sent to the user. Required if creditAmount or debitAmount are set.
creditAmount	(Optional): Set this value if the user wants a quote based on the creditAmount.
debitAmount	(Optional): Set this value if the user wants a quote based on the debitAmount.

Both debitAmount and creditAmount cannot be provided in the same request.

RESPONSE

Parameter	Description
liquidityProviders	Map of LP position IDs to Liquidity Provider.

LIQUIDITY PROVIDER

Field	Description
availableFunds	The funds available for the LP.
starkKey	The public stark key for the LP.
quote	The Liquidity Provider Quote given the user's request. Null if no request from the user or the request is unfillable by this LP.

LIQUIDITY PROVIDER QUOTE

Field	Description
creditAsset	The asset that would be sent to the user on L1.
creditAmount	The amount of creditAsset that would be sent to the user (human readable).
debitAmount	The amount of USD that would be deducted from the users L2 account (human readable).

Get Market Stats

Get Market Stats

```
from dydx3.constants import MARKET_BTC_USD

market_statistics = client.public.get_stats(
  market=MARKET_BTC_USD,
  days=MARKET_STATISTIC_DAY_SEVEN,
```

```
)
```

HTTP REQUEST

`GET v3/stats/:market`

Description: Get an individual market's statistics over a set period of time or all available periods of time.

REQUEST

Parameter	Description
market	Market whose statistics are being fetched.
days	(Optional): Specified day range for the statistics to have been compiled over. Can be one of `1`, `7`, `30`.

RESPONSE

Parameter	Description
market	The symbol of the market, e.g. ETH-USD.
open	The open price of the market.
high	The high price of the market.
low	The low price of the market.
close	The close price of the market.
baseVolume	The total amount of base asset traded.
quoteVolume	The total amount of quote asset traded.
type	Type of the market. This will always be `PERPETUAL` for now.

Get Historical Funding

Get Historical Funding

```
from dydx3.constants import MARKET_BTC_USD

historical_funding = client.public.get_historical_funding(
  market=MARKET_BTC_USD,
)
```

HTTP REQUEST

`GET v3/historical-funding/:market`

Description: Get the historical funding rates for a market.

REQUEST

Parameter	Description
market	Market whose historical funding rates are being fetched.
effectiveBeforeOrAt	(Optional): Set a date by which the historical funding rates had to be created.

RESPONSE

Parameter	Description
historicalFunding	Array of HistoricalFunding. See below for individual example.

HISTORICAL FUNDING

Parameter	Description
market	Market for which to query historical funding.
rate	The funding rate (as a 1-hour rate).
price	Oracle price used to calculate the funding rate.
effectiveAt	Time at which funding payments were exchanged at this rate.

Get Candles for Market

Get Candles for Market

```
from dydx3.constants import MARKET_BTC_USD

candles = client.public.get_candles(
  market=MARKET_BTC_USD,
  resolution='1DAY',
)
```

HTTP REQUEST

`GET v3/candles/:market`

Description: Get the candle statistics for a market.

REQUEST

Parameter	Description
market	Market whose candles are being fetched.
resolution	(Optional): Specific candle resolution being fetched. Can be one of `1DAY`, `4HOURS`, `1HOUR`, `30MINS`, `15MINS`, `5MINS`, `1MIN`.
fromISO	(Optional): Starting point for the candles.
toISO	(Optional): Ending point for the candles.
limit	(Optional): The number of candles to fetch (Max 100).

RESPONSE

Parameter	Description
startedAt	When the candle started, time of first trade in candle.
market	Market the candle is for.
resolution	Time-period of candle (currently 1HOUR or 1DAY).

Parameter	Description
low	Low trade price of the candle.
high	High trade price of the candle.
open	Open trade price of the candle.
close	Close trade price of the candle.
baseTokenVolume	Volume of trade in baseToken currency for the candle.
trades	Count of trades during the candle.
usdVolume	Volume of trade in USD for the candle.
startingOpenInterest	The open interest in baseToken at the start of the candle.

Get Global Configuration Variables

HTTP REQUEST

`GET v3/config`

Description: Get any global configuration variables for the exchange as a whole.

RESPONSE

Parameter	Description
maxFastWithdrawalAmount	The maximum amount (in USDC) allowed for fast withdrawals.
defaultMakerFee	The default maker fee for new accounts.
defaultTakerFee	The default taker fee for new accounts.
exchangeAddress	The address of the exchange contract.
collateralTokenAddress	The address of the token used as collateral.
collateralAssetId	The assetId of the collateral asset in the Starkware system.

Check If User Exists

Check If User Exists

```
user_exists = client.public.check_if_user_exists(
  ethereum_address='foo',
)
```

HTTP REQUEST

`GET v3/users/exists`

Description: Check if a user exists for a given Ethereum address.

REQUEST

Parameter	Description
ethereumAddress	Ethereum address that the user would be associated with.

RESPONSE

Parameter	Description
exists	If a user exists for the given Ethereum address.

Check If Username Exists

Check If User Exists

```
username_exists = client.public.check_if_username_exists(
  username='username',
)
```

HTTP REQUEST

`GET v3/usernames`

Description: Check if a username has been taken by a user.

REQUEST

Parameter	Description
username	Unique username being checked.

RESPONSE

Parameter	Description
exists	If a username has been taken by any user.

Get API Server Time

Get Time

```
time_object = client.public.get_time()
```

HTTP REQUEST

`GET v3/time`

Description: Get the current time of the API server.

RESPONSE

Parameter	Description

Parameter	Description
iso	ISO time of the server in UTC.
epoch	Epoch time in seconds with milliseconds.

Get Public Leaderboard PNLs

Get Leaderboard PNLs

Only available for the typescript client and http requests

HTTP REQUEST

`GET v3/leaderboard-pnl`

Description: Get the top PNLs for a specified period and how they rank against each other.

REQUEST

Parameter	Description
period	Which time period of PNLs. "DAILY" or "WEEKLY" or "MONTHLY" or "ALLTIME"
sortBy	Which PNL to sort ranks by. "ABSOLUTE" or "PERCENT"
limit	(Optional): The number of leaderboard PNLs to fetch (Max 100).

RESPONSE

Parameter	Description
username	Publically-displayed username. Placeholder username shown if not sharing
ethereumAddress	User's associated ethereum address. `null` if not sharing
absolutePnl	The PNL (in USD) for the specified period. Sorted DESC for "ABSOLUTE" sortBy
percentPnl	The percent PNL for the specified period. Sorted DESC for "PERCENT" sortBy
absoluteRank	User's absolute PNL rank
percentRank	User's percent PNL rank

Get Public Retroactive Mining Rewards

Get Public Retroactive Mining Rewards

```
rewards = client.public.get_public_retroactive_mining_rewards(
  ethereum_address='foo',
)
```

HTTP REQUEST

`GET v3/rewards/public-retroactive-mining`

Description: Get the retroactive mining rewards for an ethereum address.

REQUEST

Parameter	Description
ethereumAddress	An Ethereum address.

RESPONSE

Parameter	Description
allocation	The number of allocated dYdX tokens for the address.
targetVolume	The addresses required trade volume (in USD) to be able to claim the allocation.

V3 Websocket API

---

dYdX offers a WebSocket API for streaming v3 updates.

You can connect to the v3 WebSockets at:

- **Production**: `wss://api.dydx.exchange/v3/ws`
- **Staging (Ropsten)**: `wss://api.stage.dydx.exchange/v3/ws`

The server will send pings every 30s and expects a pong within 10s. The server does not expect pings, but will respond with a pong if sent one.

Accounts channel

This channel provides realtime information about orders, fills, funding updates and positions for a user. To subscribe, you will need to be authenticated.

To subscribe:

field	type	description
type	string	Set to `subscribe`
channel	string	Set to `v3_accounts`
accountNumber	string	The account number to subscribe to
apiKey	string	The apiKey for the user
signature	string	validation signature. See below
timestamp	string	timestamp used for the signature
passphrase	string	The `passphrase` field of the API key

Authentication

The authentication in the accounts channel is identical to private endpoint authentication with one key difference. The `requestPath` is `/ws/accounts`.

INITIAL RESPONSE:

The initial response will contain the information about the account, open positions, recent transfers, and open orders, i.e. everything from GET `/v3/accounts/:id`, GET `/v3/transfers`, GET `/v3/funding` and GET `/v3/orders` (with `accountId` in the header).

Note that the `freeCollateral` and `equity` (also called `total account value`) for an account are only sent in the initial response. To track these over time, refer to this section.

> Example initial response

CHANNEL DATA

Subsequent responses will contain any updates to open orders, or changes to account balance, or the open positions, or transfers, in a single message.

> A fill occurs, and a position is closed, and the account balance modified

> a deposit occurs

Orderbook

To subscribe:

field	type	description
type	string	Set to `subscribe`
channel	string	Set to `v3_orderbook`
id	string	The market to subscribe to e.g. BTC-USD, LINK-USD
includeOffsets (optional)	boolean	If specified, this will return an initial response with per-price level offsets

INITIAL RESPONSE:

The initial response will contain the state of the orderbook and will be the same structure as GET `/v3/orderbook/:market`. If `includeOffsets` is sent and set to true in the subscription message, there will be an offset included for each price level. (See the example included)

field	description
type	will be `subscribed`
channel	the channel name, i.e. `v3_orderbook`
id	the market subscribed to e.g. BTC-USD
contents	the message contents

The contents is structured as:

field	type	description
offset	string	A number used for ordering. See `offset` below.
bids	array	See `PublicOrder` below. Sorted by price in descending order

field	type	description
asks	array	See `PublicOrder` below. Sorted by price in ascending order

PublicOrder:

field	type	description
price	string	human readable price of the order (in quote / base currency)
size	string	human readable size of the order (in base currency)
offset	string	(if `includeOffsets` is set to true) the offset for the specific price

Offset:

The price updates are not guaranteed to be sent in order. So it is possible to receive an older price update later. For this reason, the offset is included in the message, to help order. The offset increases monotonically, and increasing values of offsets indicate more recent values.

> To keep a valid orderbook, you should store the offset for each price level independently. A given price level should be updated if and only if an update for the price level is received with a higher offset than what you have stored. To get a per-price level offset in the initial response, you can set `includeOffsets` to true when subscribing.

Example messages:

> Example initial response:
>
> Example initial response if `includeOffsets` is set to true:

Request:

Response:

CHANNEL DATA

Subsequent responses will contain the new order sizes for any price levels that have changed since the previous update:

e.g:

> Subsequent messages

E.g: if some orders at "102" price, get filled, then the update would be ["102", "12"], where "12" is the new size. If there are no more asks at "104", then the ask update would be ["104", "0"].

Trades

To subscribe:

field	type	description
type	string	Set to `subscribe`
channel	string	Set to `v3_trades`
id	string	The market to subscribe to e.g. BTC-USD, LINK-USD

INITIAL RESPONSE:

The initial response will contain the historical trades and will be the same structure as GET `/v3/trades/:market`.

field	description
type	will be `subscribed`
channel	the channel name, i.e. `v3_trades`
id	the market subscribed to e.g. BTC-USD
contents	the message contents

The contents is structured as:

field	type	description
trades	array	See `PublicTrade` below.

PublicTrade:

field	type	description
side	string	`BUY` or `SELL`
size	string	size of the trade
price	string	price of the trade
createdAt	ISO time of the trade	time of the trade

Example messages:

> Example initial response:

CHANNEL DATA

Subsequent responses will contain the recently created trades. e.g:

> Subsequent responses

Markets

To subscribe:

field	type	description
type	string	Set to `subscribe`
channel	string	Set to `v3_markets`

INITIAL RESPONSE:

Same as GET /v3/markets

CHANNEL DATA

Subsequent responses will contain an update for one or more markets. Updates will be sent any time a field(s) changes on a

market(s). Updates will only contain the field(s) that have changed:

Subsequent responses

Security

Independent Audits

The Starkware Perpetual smart contracts were audited independently by PeckShield.

PeckShield Audit Report

Vulnerability Disclosure Policy

The disclosure of security vulnerabilities helps us ensure the security of our users.

How to report a security vulnerability?

If you believe you've found a security vulnerability in one of our contracts or platforms, send it to us by emailing security@dydx.exchange. Please include the following details with your report:

- A description of the location and potential impact of the vulnerability.
- A detailed description of the steps required to reproduce the vulnerability.

Scope

Any vulnerability not previously disclosed by us or our independent auditors in their reports.

Guidelines

We require that all reporters:

- Make every effort to avoid privacy violations, degradation of user experience, disruption to production systems, and destruction of data during security testing.
- Use the identified communication channels to report vulnerability information to us.
- Keep information about any vulnerabilities you've discovered confidential between yourself and dYdX until we've had 30 days to resolve the issue.

If you follow these guidelines when reporting an issue to us, we commit to:

- Not pursue or support any legal action related to your findings.
- Work with you to understand and resolve the issue quickly (including an initial confirmation of your report within 72 hours of submission).
- Grant a monetary reward based on the OWASP risk assessment methodology.