Validator Client

Getting Started

Installation

TypeScript Python pnpm install

@dydxprotocol/v4-client-js

Initializing the Client

TypeScript Python import { ValidatorClient , Network } from

"@dydxprotocol/v4-client-js" ;

/** // For the deployment by DYDX token holders, use below:

import { IndexerConfig, ValidatorConfig } from "@dydxprotocol/v4-client-js";

const NETWORK: Network = new Network('mainnet', new IndexerConfig('https://indexer.dydx.trade', 'wss://indexer.dydx.trade',), new ValidatorConfig('https://dydx-ops-rpc.kingnodes.com', // or other node URL 'dydx-mainnet-1', { CHAINTOKEN_DENOM: 'adydx', CHAINTOKEN_DECIMALS: 18, USDC_DENOM: 'ibc/8E27BA2D5493AF5636760E354E46004562C46AB7EC0CC4C1CA14E9E20E2545B5', USDC_GAS_DENOM: 'uusdc', USDC_DECIMALS: 6, },),); */ const

NETWORK

=

Network .testnet ();

const

client

=

await

ValidatorClient .connect (NETWORK .validatorConfig);

Configuring a Network

TypeScript import { Network , ValidatorClient , IndexerConfig , ValidatorConfig } from

'@dydxprotocol/v4-client-js' ;

const

indexerConfig

=

new

IndexerConfig ({INDEXER_REST_URL} , {INDEXER_WEBSOCKET_URL});

const

denomConfig

= { USDC_DENOM :

'ibc/8E27BA2D5493AF5636760E354E46004562C46AB7EC0CC4C1CA14E9E20E2545B5' , USDC_DECIMALS :

6 , USDC_GAS_DENOM :

'uusdc' , CHAINTOKEN_DENOM : {CHAIN_TOKEN_DENOM} //string CHAINTOKEN_DECIMALS : {CHAIN_TOKEN_DECIMALS} //integer };

const

validatorConfig

=

new

ValidatorConfig ({VALIDATOR_REST_URL} , {CHAIN_ID} , denomConfig);

const

custom_network

=

new

Network ('custom-network-name' , indexerConfig , validatorConfig);

const

client

=

await

ValidatorClient .connect (custom_network .validatorConfig);

Creating a LocalWallet

TypeScript Python import { BECH32_PREFIX , LocalWallet , } from

'@dydxprotocol/v4-client-js' ;

const

mnemonic

=

'YOUR MNEMONIC HERE' ; const

wallet

=

await

LocalWallet .fromMnemonic (mnemonic ,

BECH32_PREFIX);

Simulate, Sign and Send Transactions

Simulate a Transaction

TypeScript Python const

messages

= () =>

Promise .resolve ([/ *... your transaction messages here*/]); const

fee

=

await

client .simulate (wallet , messages);

Sign a Transaction

TypeScript Python const

messages

= () =>

Promise .resolve ([/ *... your transaction messages here*/]); const

zeroFee

=

true ; const

signedTransaction

=

await

client .sign (wallet , messages , zeroFee);

Send a Transaction

TypeScript Python const

messages

= () =>

Promise .resolve ([/ *... your transaction messages here*/]); const

zeroFee

=

true ; const

signedTransaction

=

await

client .send (wallet , messages , zeroFee);

Get Account Balances

TypeScript Python // Get all balances for an account. const

balances

=

await

client . get .getAccountBalances (DYDX_ADDRESS)

// Get balance of one denom for an account. const

balance

=

await

client . get .getAccountBalance (DYDX_ADDRESS , TOKEN_DENOM)

Transfers, Deposits, and Withdraws

Transfering an Asset

TypeScript Python import { SubaccountClient } from

'@dydxprotocol/v4-client-js' ;

const

subaccount

=

new

SubaccountClient (wallet ,

0); const

recipientAddress

=

'dydx...'

// address of the recipient const

recipientSubaccountNumber

=

0

// subaccount number of the recipient const

assetId

=

0

// asset id of the token you want to transfer const

amount

=

Long .fromNumber (/ *amount of the token you want to transfer*/);

const

tx

=

await

client . post .transfer (subaccount , recipientAddress , recipientSubaccountNumber , assetId , amount);

Depositing from wallet to Subaccount

TypeScript Python import { SubaccountClient } from

'@dydxprotocol/v4-client-js' ;

const

subaccount

=

new

SubaccountClient (wallet ,

0); const

assetId

=

0

// asset id of the token you want to deposit const

amount

=

Long .fromNumber (/ *amount of the token you want to deposit*/);

const

tx

=

await

client . post .deposit (subaccount , assetId , amount);

Withdrawing from Subaccount to wallet

TypeScript Python import { SubaccountClient } from

'@dydxprotocol/v4-client-js' ;

const

subaccount

=

new

SubaccountClient (wallet ,

0); const

assetId

=

0

// asset id of the token you want to withdraw const

amount

=

Long .fromNumber (/ *amount of the token you want to withdraw*/);

const

tx

=

await

client . post .withdraw (subaccount , assetId , amount);

Placing and Cancelling Orders

Placing an Order

TypeScript Python import { OrderFlags , Order_Side , Order_TimeInForce , SubaccountClient } from

'@dydxprotocol/v4-client-js' ;

const

subaccount

=

new

SubaccountClient (wallet ,

0); const

clientId

=

123

// set to a number, can be used by the client to identify the order const

clobPairId

=

0

// perpertual market id const

side

=

Order_Side . SIDE_BUY

// side of the order const

quantums

=

Long .fromNumber (1_000_000_000); // quantums are calculated by the size if the order const

subticks

=

Long .fromNumber (1_000_000_000); // subticks are calculated by the price of the order const

timeInForce

=

Order_TimeInForce . TIME_IN_FORCE_UNSPECIFIED ; // TimeInForce indicates how long an order will remain active before it is executed or expires const

orderFlags

=

OrderFlags . SHORT_TERM ; // either SHORT_TERM, LONG_TERM or CONDITIONAL const

reduceOnly

=

false ; // if true, the order will only reduce the position size

const

tx

=

await

client . post .placeOrder (subaccount , clientId , clobPairId , side , quantums , subticks , timeInForce , orderFlags , reduceOnly);

Setting the good-til-block

When specifying the good-til-block on your order, verify that the following is true to ensure your order placement succeeds (whereShortBlockWindow is currently set to[20 blocks(opens in a new tab)](#)):

currentBlockHeight < order.goodTilBlock <= currentBlockHeight + ShortBlockWindow .

Replacing an Order

Traders can replace Short-Term orders atomically by placing an order with the same order ID and a larger value for the[good-til-block field(opens in a new tab)](#) of the order.

Note that two orders have the same order ID if the following client-specified fields are equal (from[OrderId proto definition(opens in a new tab)](#)):

- [Subaccount ID(opens in a new tab)](#)
- .* order.subaccount_id.owner should be set to your address that is signing the order transaction.
-
    - order.subaccount_id.number should be set to 0 unless you are using a different subaccount.
- Client ID.
- Order flags (note this should always be set to 0 for placing Short-Term orders).
- CLOB pair ID.

Assuming the current block height is 9, the below example places an order with good-til-block 10, then places a replacement order with a good-til-block of 11.

TypeScript Python import { OrderFlags , Order_Side , Order_TimeInForce , SubaccountClient } from

'@dydxprotocol/v4-client-js' ;

const

subaccount

=

new

SubaccountClient (wallet ,

0); const

clientId

=

123

// set to a number, can be used by the client to identify the order const

clobPairId

=

0

// perpertual market id const

side

=

Order_Side . SIDE_BUY

// side of the order const

quantums

=

Long .fromNumber (1_000_000_000); // quantums are calculated by the size if the order const

subticks

=

Long .fromNumber (1_000_000_000); // subticks are calculated by the price of the order const

timeInForce

=

Order_TimeInForce . TIME_IN_FORCE_UNSPECIFIED ; // TimeInForce indicates how long an order will remain active before it is executed or expires const

orderFlags

=

OrderFlags . SHORT_TERM ; // either SHORT_TERM, LONG_TERM or CONDITIONAL const

reduceOnly

=

false ; // if true, the order will only reduce the position size

const

tx

=

await

client . post .placeOrder (subaccount , clientId , clobPairId , side , quantums , subticks , timeInForce , orderFlags , reduceOnly , 10 ,);

const

replacementTx

=

await

client . post .placeOrder (subaccount , clientId , clobPairId , side , quantums , subticks , timeInForce , orderFlags , reduceOnly , 11 ,); As of February 23rd, 2024, Typescript client source code for the above function ishere(opens in a new tab) , and Python client source code for the above function ishere(opens in a new tab) .

Cancelling an Order

All paramsters are fromOrder object from indexer goodTilBlockTime is the UTC epoch second of the order's goodTilBlockTime One and only one of goodTilBlock and goodTilBlockTime should be passed in as a parameter

TypeScript Python / *order is an Order object from the Indexer*/ const

goodTilBlock

=

order .goodTilBlock let goodTilBlockTime :

```
number

|

undefined ; if ( order .goodTilBlockTime) { const

datetime

=

new

Date ( order .goodTilBlockTime); const

utcMilllisecondsSinceEpoch

=

datetime .getTime () goodTilBlockTime =

Math .round (utcMilllisecondsSinceEpoch /

1000 ); }

const

tx

=

await

client . post .cancelOrder ( subaccount , order .clientId , order .orderFlags , order .clobPairId , goodTilBlock ,
goodTilBlockTime );
```
Last updated onFebruary 29, 2024 Socket Client Composite Client