

Composite Client

Getting Started

CompositeClient simplifies the transactions by transforming human readable parameters to chain-specific parameters.

For example, Placing an order with ValidatorClient requires parameters such as quantums and subticks, which are calculated based on ClobPair settings.

CompositeClient would take regular human readable parameters such as price and size, make the necessary calculations and conversions, and call ValidatorClient with the correct quantums and subticks for you.

Installation

TypeScript Python npm install

@dydxprotocol/v4-client-js

Initializing the Client

TypeScript Python import { CompositeClient , 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

CompositeClient .connect (NETWORK);

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 );
```

Placing Orders

Placing an Order

```
TypeScript Python import { OrderExecution , OrderSide , OrderTimeInForce , OrderType , } 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  
market  
=  
"BTC-USD" ; // perpertual market id const  
type  
=  
OrderType . LIMIT ; // order type const  
side  
=  
OrderSide . BUY ; // side of the order const  
timeInForce  
=  
OrderTimeInForce . IOC ; // UX TimeInForce const  
execution  
=  
OrderExecution . DEFAULT ; const  
price  
=  
30_000 ; // price of 30,000; const  
size  
=  
0.1 ; // subticks are calculated by the price of the order const  
postOnly  
=  

```

```

false ; // If true, order is post only const
reduceOnly

=

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

=

null ; // required for conditional orders

const

tx

=

await

client .placeOrder ( subaccount , market , type , side , price , size , clientId , timeInForce , 0 , execution , postOnly ,
reduceOnly , triggerPrice );

```

Replacing an Order

For more details on order replacements, see [Replacing an Order](#) in the validator client section.

Canceling an Order

```

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

tx

=

await

client .cancelOrder ( subaccount , order .clientId , order .orderFlags , order .clobPairId , order .goodTilBlock , order
.goodTilBlockTime ); Last updated on May 30, 2024 Validator Client Indexer API

```