Order creation examples

Through@1inch/limit-order-protocol-utils

This is the recomended option! There isJavaScript and Typescript compatible library that solves everything for you, we taking care on it to make everything up to date and make you migration over Limit Order Protocol Versions as smooth as possible.

- Seedocs on order creation
- Seelibrary docs

Python Example for 1inch Limit Order v3

To place a limit order in python requires more work since there's no library

In this example the predicate only uses the timestamp to determine when the order should expire but there are many more functions that can be used for the predicate. See thelimit order protocol utils github for more information.

from eth_account . messages import encode_structured_data from web3 import Web3 import requests import time w3 = Web3 (Web3 . HTTPProvider ("https://cloudflare-eth.com"))

you can customize the RPC

wallet_key

"..."

Your wallet private key without the leading 0x

wallet_address

"..."

Your wallet address

limit_order_contract

"0x1111111254EEB25477B68fb85Ed929f73A960582"

the limit order contract (now the same as the 1inch v5 router)

chain_id

1

the chain id of the network you are using ##didn't exist in the previoius version

ETHERSCAN_API_KEY

"yourapikeytoken"

Etherscan API key, this may not be required or should be changed if the ABIs are changed to literals or a different blockchain API is used like api.bscscan.com or api.polygonscan.com

create the limit order contract instance

limit_order_contract_abi_response

requests . get (f"https://api.etherscan.io/api?module=contract&action=getabi&address= { limit_order_contract } &apikey= { ETHERSCAN_API_KEY } ") limit_order_contract_abi = limit_order_contract_abi_response . json ( ) ["result"] limit_order_contract_instance = w3 . eth . contract (address = limit_order_contract , abi = limit_order_contract_abi)

wait 5 seconds to avoid rate limiting

time . sleep (5)

get the token addresses for the tokens you want to trade

erc20_abi_response

requests . get (f"https://api.etherscan.io/api?module=contract&action=getabi&address=0x6b175474e89094c44da98b954eedeac495271d0f&apikey= { ETHERSCAN_API_KEY } ") erc20_abi = erc20_abi_response . json ( ) ["result"]

here is were we define parameters for the limit order

makerAddress

Web3 . toChecksumAddress (wallet_address)

the address of the wallet that will be the maker of the order

takerAddress

Web3 . toChecksumAddress ("0x0000000000000000000000000000000000000000")

the address of the taker, if it's address(0) then it's a public order

makerAsset

Web3 . toChecksumAddress ("0x6b175474e89094c44da98b954eedeac495271d0f")

the address of the token you want to sell

takerAsset

Web3 . toChecksumAddress ("0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2")

the address of the token you want to buy

makerAmount

1000

the amount of the token you want to sell in wei

takerAmount

1000000000000000000

the amount of the token you want to buy in wei

makerAssetContract

w3 . eth . contract (address = makerAsset , abi = erc20_abi) takerAssetContract = w3 . eth . contract (address = takerAsset , abi = erc20_abi)

other order parameters

makerAssetData

'0x' takerAssetData =

'0x' getMakingAmount =

'0x' getTakingAmount =

'0x' expiration =

5444440000

some time in the future

nonce

0

the nonce of the order, used to be able to cancel all orders that have the same nonce by increasing the addresses' nonce

seriesNonceManagerContractAddress

w3 . toChecksumAddress ('0x303389f541ff2d620e42832f180a08e767b28e10')

https://github.com/1inch/limit-order-protocol-utils/blob/fdbb559509eeb6e22e2697cccb22887d69617652/src/series-nonce-manager.const.ts

seriesNonceManagerABI_response

requests . get (f"https://api.etherscan.io/api?module=contract&action=getabi&address= { seriesNonceManagerContractAddress } &apikey= { ETHERSCAN_API_KEY } ") seriesNonceManagerABI =

[{ "inputs" : [ ] , "name" : "AdvanceNonceFailed" , "type" : "error" } , { "anonymous" : False , "inputs" : [{ "indexed" : True , "internalType" : "address" , "name" : "maker" , "type" : "address" } , { "indexed" : False , "internalType" : "uint256" , "name" : "series" , "type" : "uint256" } , { "indexed" : False , "internalType" : "uint256" , "name" : "newNonce" , "type" : "uint256" }] , "name" : "NonceIncreased" , "type" : "event" } , { "inputs" : [{ "internalType" : "uint256" , "name" : "series" , "type" : "uint256" } , { "internalType" : "uint256" , "name" : "amount" , "type" : "uint256" }] , "name" : "advanceNonce" , "outputs" : [ ] , "stateMutability" : "nonpayable" , "type" : "function" } , { "inputs" : [{ "internalType" : "uint8" , "name" : "series" , "type" : "uint8" }] , "name" : "increaseNonce" , "outputs" : [ ] , "stateMutability" : "nonpayable" , "type" : "function" } , { "inputs" : [{ "internalType" : "uint256" , "name" : "" , "type" : "uint256" } , { "internalType" : "address" , "name" : "" , "type" : "address" }] , "name" : "nonce" , "outputs" : [{ "internalType" : "uint256" , "name" : "" , "type" : "uint256" }] , "stateMutability" : "view" , "type" : "function" } , { "inputs" : [{ "internalType" : "uint256" , "name" : "series" , "type" : "uint256" } , { "internalType" : "address" , "name" : "makerAddress" , "type" : "address" } , { "internalType" : "uint256" , "name" : "makerNonce" , "type" : "uint256" }] , "name" : "nonceEquals" , "outputs" : [{ "internalType" : "bool" , "name" : "" , "type" : "bool" }] , "stateMutability" : "view" , "type" : "function" } , { "inputs" : [{ "internalType" : "uint256" , "name" : "time" , "type" : "uint256" }] , "name" : "timestampBelow" , "outputs" : [{ "internalType" : "bool" , "name" : "" , "type" : "bool" }] , "stateMutability" : "view" , "type" : "function" } , { "inputs" : [{ "internalType" : "uint256" , "name" : "timeNonceSeriesAccount" , "type" : "uint256" }] , "name" : "timestampBelowAndNonceEquals" , "outputs" : [{ "internalType" : "bool" , "name" : "" , "type" : "bool" }] , "stateMutability" : "view" , "type" : "function" }] seriesNonceManagerInstance = w3 . eth . contract (address = seriesNonceManagerContractAddress , abi = seriesNonceManagerABI) One of the breaking changes with v3 was the addition of the offsets parameter and the packing of the interactions

this data will be all packed into the Interactions parameter

series

0

0 is limit order 1 is p2p order

nonceManagerCalldata

seriesNonceManagerInstance . encodeABI (fn_name = "timestampBelow" , args = [expiration]) predicate = limit_order_contract_instance . encodeABI (fn_name = "arbitraryStaticCall" , args = [seriesNonceManagerContractAddress , nonceManagerCalldata]

) permit =

'0x'

this would be used to add an EIP 712 permit to the order

preInteraction

'0x'

this would be used to add a pre-interaction to the order

postInteraction

'0x'

this would be used to add a post-interaction to the order, for example unwrapping wETH to ETH

all_interactions

[makerAssetData , takerAssetData , getMakingAmount , getTakingAmount , predicate , permit , preInteraction , postInteraction]

this function will calculate the offsets for the interactions

def

getOffsets (interactions) : lenghtMap =

[ ] for interaction in interactions : if interaction [0 : 2]

==

"0x" : lenghtMap . append (int (len (interaction) / 2

-

1)) else : lenghtMap . append (int (len (interaction) / 2)) cumulativeSum =

0 bytesAccumularot =

0 index =

0 UINT32_BITS =

32

print(lenghtMap)

for lenght in lenghtMap : cumulativeSum += lenght

bytesAccumularot + (BigInt(offset) << ((exports.UINT32_BITS * BigInt(index))));

print(str(bytesAccumularot) + '' + ('' + str(cumulativeSum) + '' << ('' + str(UINT32_BITS) + '' * '' + str(index) + ''))'')

bytesAccumularot += cumulativeSum <<

(UINT32_BITS * index) index +=

1 offsets = bytesAccumularot

hex(bytesAccumularot)

return offsets

offsets

getOffsets (all_interactions)

def

trim0x (hexString) : if hexString [0 : 2]

==

'0x' : return hexString [2 :] return hexString

interactions

"0x" for interaction in all_interactions : interactions = interactions + trim0x (interaction) a limit order can't be signed properly if all the types are strings here we define what each field is and the type it should be

order_data

{ "salt" :

0 , "makerAsset" : makerAsset , "takerAsset" : takerAsset , "maker" : makerAddress , "receiver" : takerAddress , "allowedSender" :

"0x0000000000000000000000000000000000000000" , "makingAmount" : makingAmount , "takingAmount" : takingAmount , "offsets" : offsets , "interactions" : interactions }

order_types

[{ "name" :

"salt" ,

"type" :

"uint256" } , { "name" :

"makerAsset" ,

"type" :

"address" } , { "name" :

"takerAsset" ,

"type" :

"address" } , { "name" :

"maker" ,

"type" :

"address" } , { "name" :

"receiver" ,

"type" :

"address" } , { "name" :

"allowedSender" ,

"type" :

"address" } , { "name" :

"makingAmount" ,

"type" :

"uint256" } , { "name" :

"takingAmount" ,

"type" :

"uint256" } , { "name" :

"offsets" ,

"type" :

"uint256" } , { "name" :

"interactions" ,

"type" :

"bytes" } ,]

this function will fix the order_data to be a typed object instead of only strings

def

fix_data_types (data , types) : """ Order data values are all strings as this is what the API expects. This function fixes their types for encoding purposes. """ fixed_data =

{ } for dictionary in types : if

"bytes"

in dictionary ["type"] : fixed_data [dictionary ["name"]]

=

(Web3 . toBytes (hexstr = data [dictionary ["name"]])) elif

"int"

in dictionary ["type"] : fixed_data [dictionary ["name"]]

=

int (data [dictionary ["name"]]) else :

address

fixed_data [dictionary ["name"]]

= data [dictionary ["name"]] return fixed_data

this is a typed data that the private key will be signing, it conforms to the EIP 712 standard

eip712_data

{ "primaryType" :

"Order" , "types" :

{ "EIP712Domain" :

[{ "name" :

"name" ,

"type" :

"string" } , { "name" :

"version" ,

"type" :

"string" } , { "name" :

"chainId" ,

"type" :

"uint256" } , { "name" :

"verifyingContract" ,

"type" :

"address" } ,] , "Order" : order_types } , "domain" :

{ "name" :

"1inch Aggregation Router" , "version" :

"5" , "chainId" : chain_id , "verifyingContract" :

"0x1111111254eeb25477b68fb85ed929f73a960582" , } , "message" : fix_data_types (order_data , order_types) , } Finally the private key will be used to sign the encoded data and broadcast to the API

If you are having problems signing please run the following commands:pip install web3 --upgrade; pip install eth-account==0.6.1

encoded_message

encode_structured_data (eip712_data) signed_message = w3 . eth . account . sign_message (encoded_message , wallet_key)

this is the limit order that will be broadcast to the limit order API

limit_order

{ "orderHash" : signed_message . messageHash . hex ( ) , "signature" : signed_message . signature . hex ( ) , "data" : order_data , }

limit_order_url

"https://limit-orders.1inch.io/v3.0/" +

str (chain_id)

+ "/limit-order"

make sure to change the chain_id if you are not using ETH mainnet

response

requests . post (url = limit_order_url , headers = { "accept" :

"application/json, text/plain, /" ,

"content-type" :

"application/json" } , json = limit_order)

print the full response

print (response . text)Edit this page Previous Limit vs. RFQ Orders Next About