How to use unregistered pools

author: yugure#3291 (Discord), @crypto2real (Twitter)

created: 2022/5/27

modified: 2022/5/27

ATTENTION) This guide is for Orca NORMAL pool (not Whirlpool)

ATTENTION) This approach is for NON-stable pair

The process is as follows.

1. recognizing the address of pool account
2. analyzing the pool account
3. defining OrcaPoolParams
4. making OrcaPoolImpl instance without calling getPool()

I will describe how to use SAMO/USDC pool without calling getPool(). Of course, SAMO/USDC pool have been registered in SDK, this description can be applied to other pools, even if pools are brand new.

1. Recognizing the address of pool account

To recognize the address of pool account, there is 2 ways.

1. picking up from allPools
2. executing swap by your self and picking up from Solscan

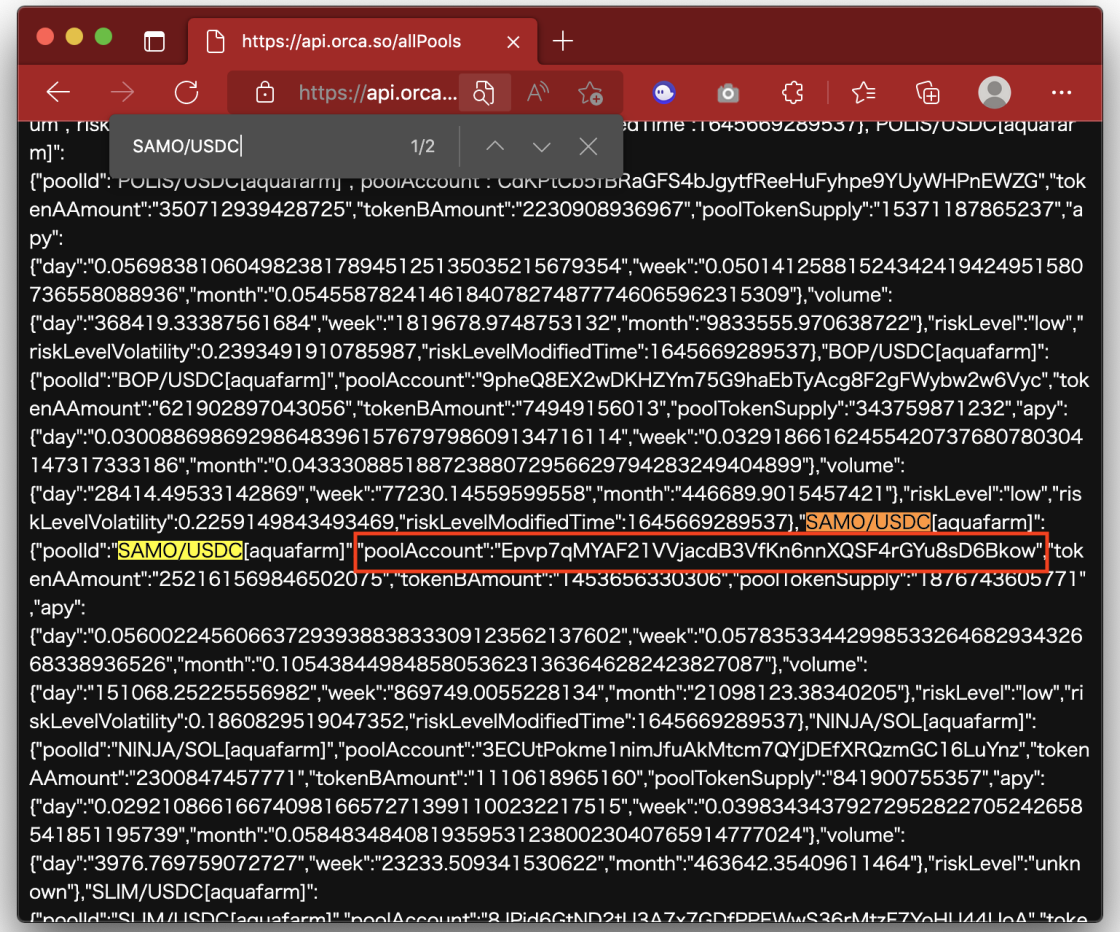
1.1. piking up from allPools

1. access to allPools (https://api.orca.so/allPools)

2. find the name of Pool (SAMO/USDC)

3. recognize poolAccount

poolAccount of SAMO/USDC is **Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow**

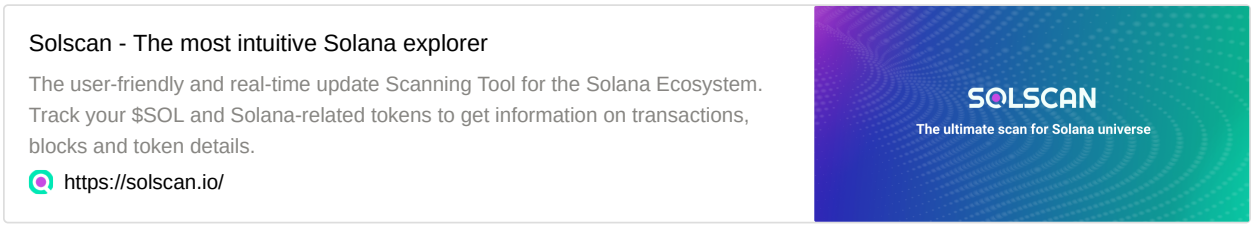


1.2. piking up from Solscan

1. execute swapping from UI with pool which you want to use

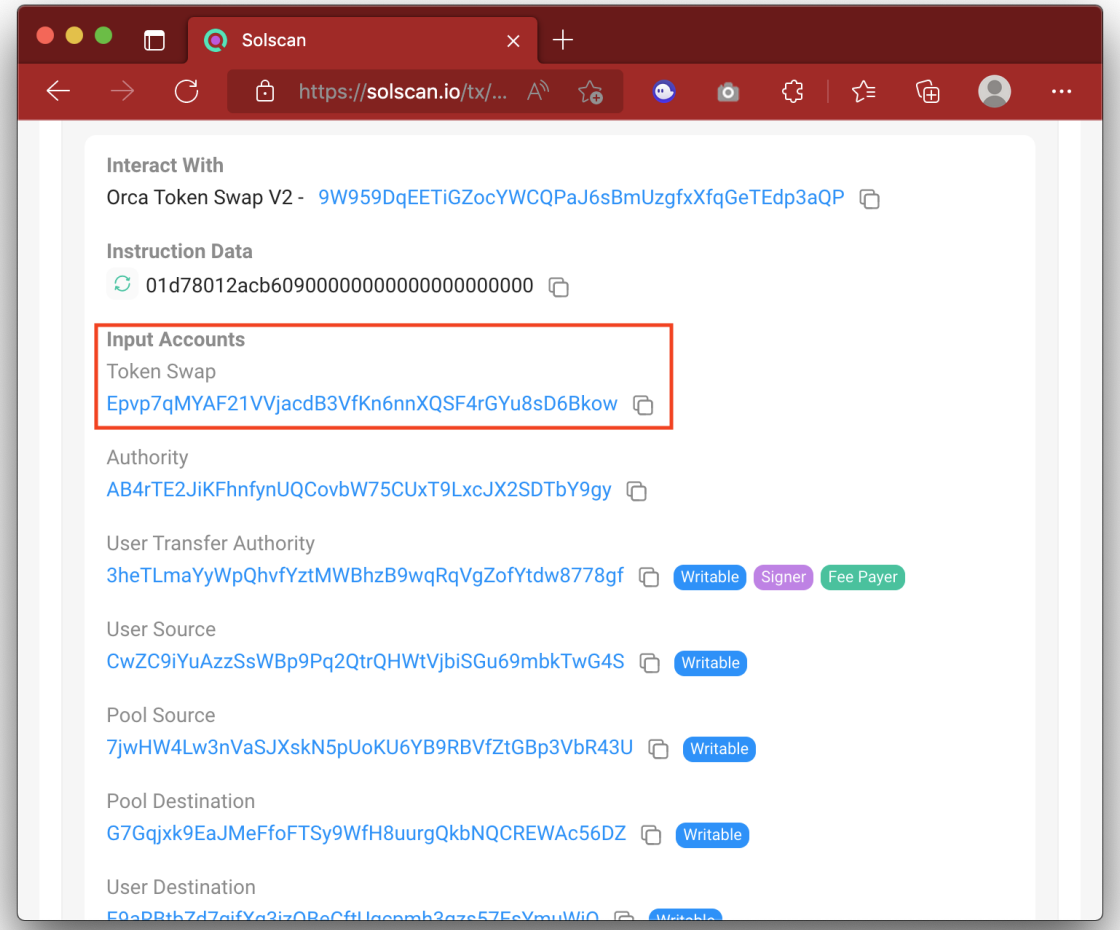
2. get Tx signature from wallet

3. enter Tx signature into Solscan



4. recognize the account shown as Token Swap

Token Swap account is **Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow**



2. Analyzing pool account

To analyze pool account, we use SOL/BORSH Decorder.

This is awesome tool.

Inspect any Solana account | Solana x Borsh Decoder

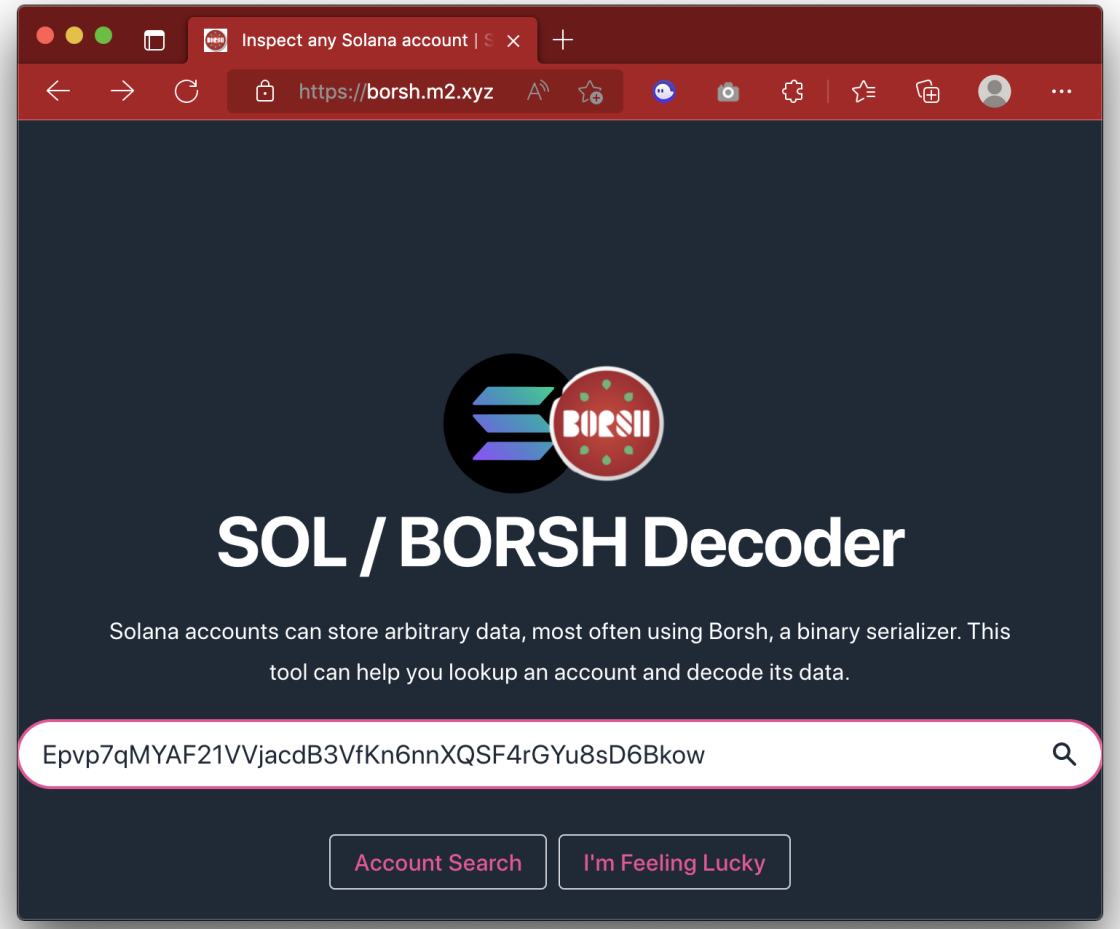
A simple online tool to help you decode borsh encoded data from Solana accounts

https://borsh.m2.xyz/

1. access to SOL/BORSH Decorder

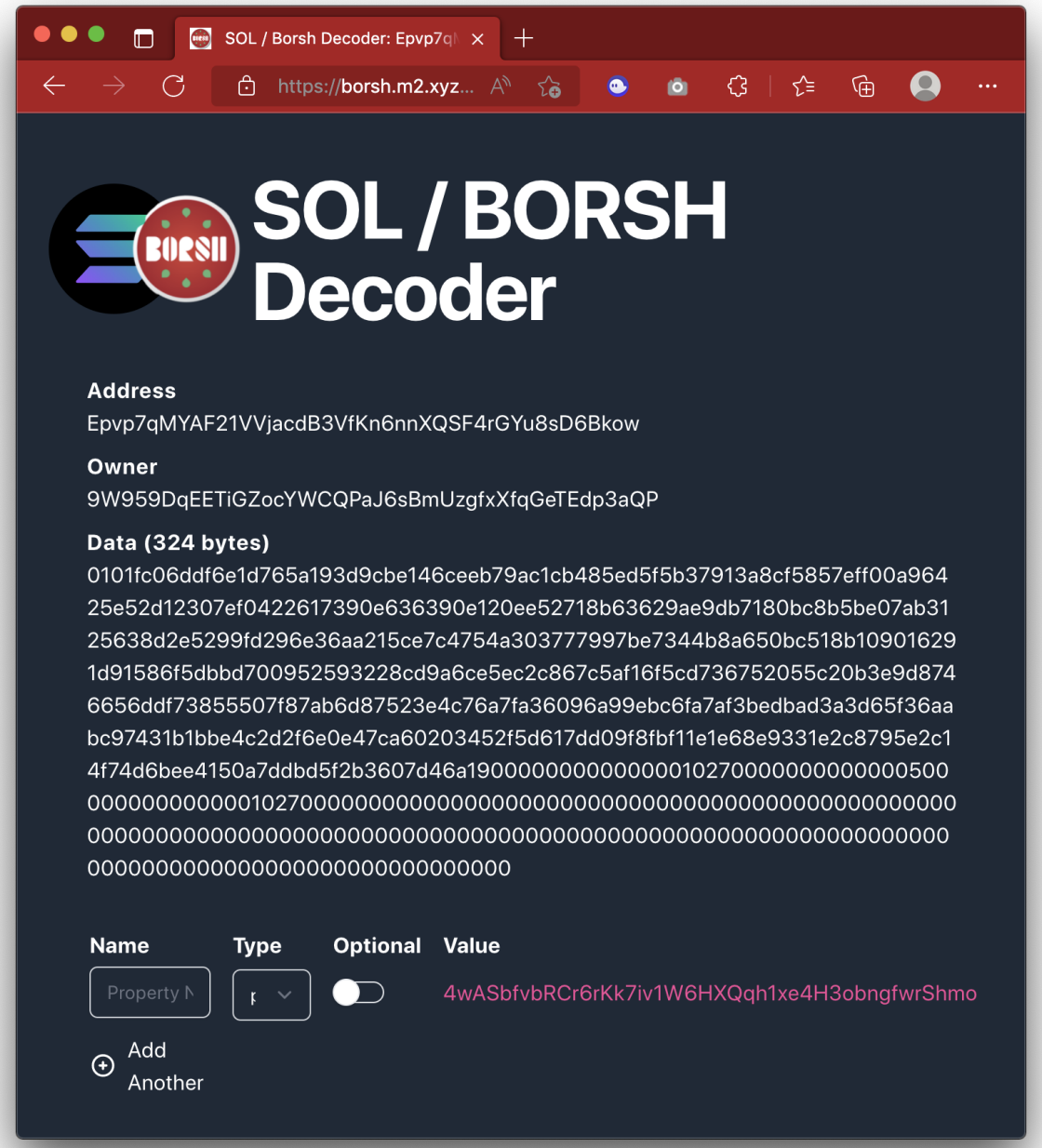
2. enter the address of pool, then click “Account Search”

The address is **Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow** (SAMO/USDC)



3. confirm that the data is fetched and shown

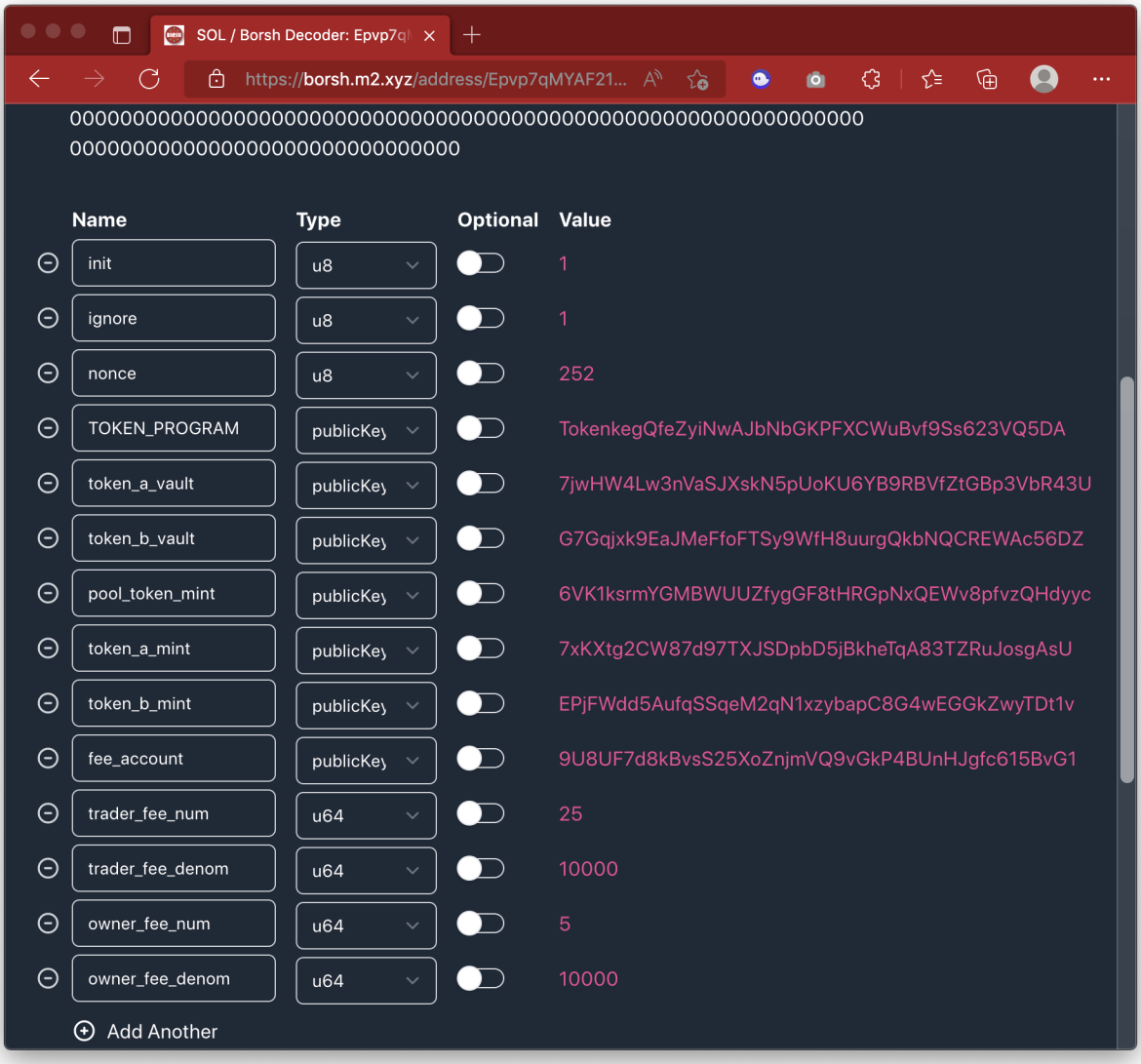
The size of account shold be 324 bytes.



4. decode account data

Add rows with “Add Another”, and input “Name”, “Type” fields as follows.

Then the tool show the human readable values.



You can confirm the correctness of this analysis with comparing to the definition of SAMO/USDC pool in SDK.

typescript-sdk/pools.ts at main · orca-so/typescript-sdk

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters. Learn more about bidirectional Unicode

https://github.com/orca-so/typescript-sdk/blob/main/src/constants/pools.ts#L888

orca-so/**typescript-sdk**

The Orca SDK contains a set of simple to use APIs to allow developers to integrate with the Orca exchange...

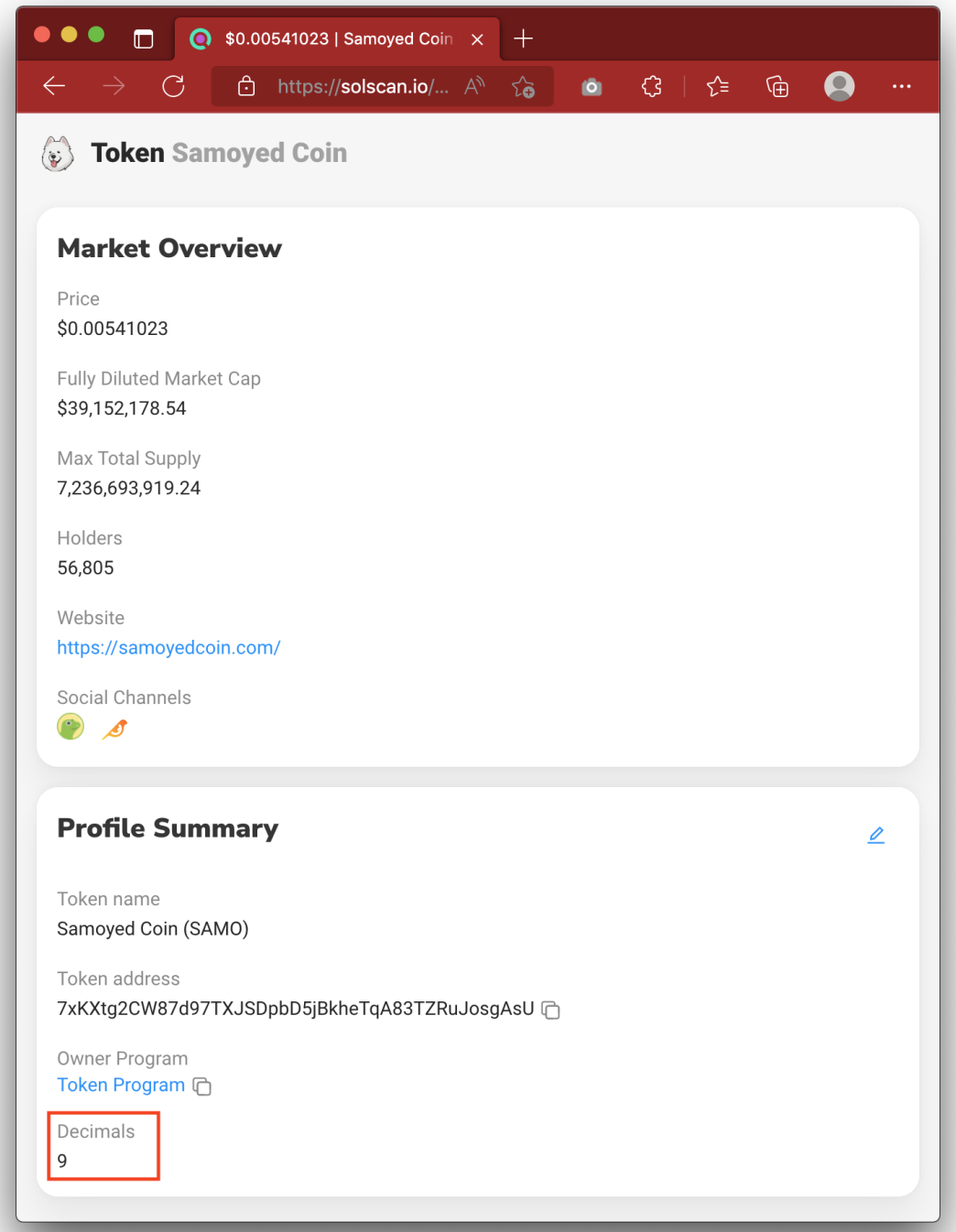
8 Contributors · 57 Used by · 115 Stars · 34 Forks

5. get token info

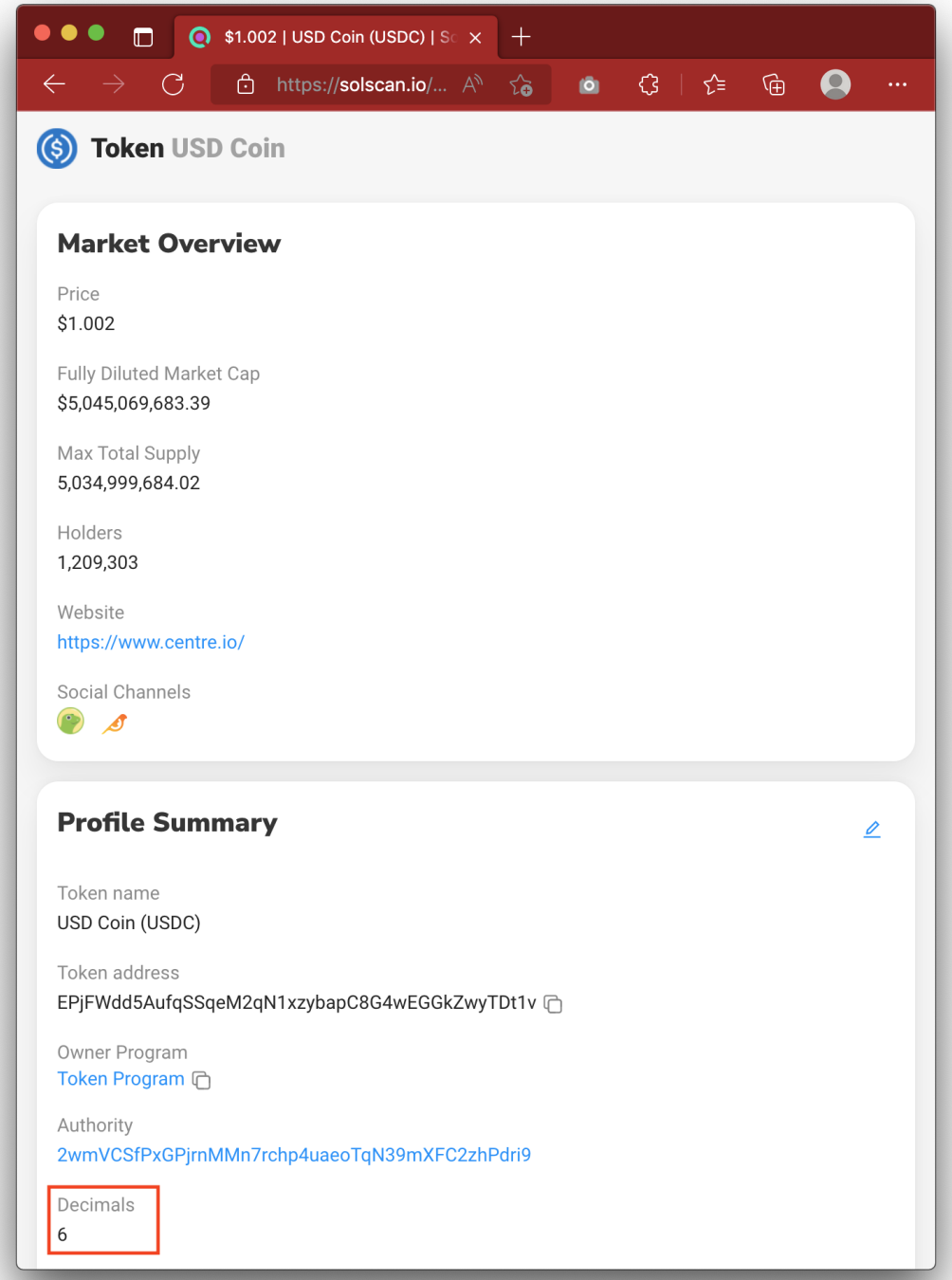
We need to get some info of tokens. Now we know the mint address of tokens, so enter them into Solscan.

Please memo the decimals of each tokens. (SAMO: 9, USDC: 6)

SAMO(token_a_mint): 7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU



USDC(token_b_mint): EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v



3. Defining OrcaPoolParams

Use the following template, and replace UPPERCASE CHARACTERS (other than ORCA_TOKEN_SWAP_ID). We have all required info already.

TEMPLATE

```
import { PublicKey, Connection } from "@solana/web3.js";
import { ORCA_TOKEN_SWAP_ID, Percentage, Network } from "@orca-so/sdk";
import { OrcaPoolParams, CurveType } from '@orca-so/sdk/dist/model/orca/pool/pool-types';
import { OrcaPoolImpl } from '@orca-so/sdk/dist/model/orca/pool/orca-pool';
import Decimal from 'decimal.js';

async function main() {
  const token_a = {
    tag: "TOKEN_A_SYMBOL",
    name: "TOKEN_A_NAME",
    mint: new PublicKey("TOKEN_A_MINT"),
    scale: TOKEN_A_DECIMALS
  };
  const token_b = {
    tag: "TOKEN_B_SYMBOL",
    name: "TOKEN_B_NAME",
    mint: new PublicKey("TOKEN_B_MINT"),
    scale: TOKEN_B_DECIMALS
  };

  const TOKENA_TOKENB_params: OrcaPoolParams = Object.freeze({
    address: new PublicKey("ADDRESS_OF_POOL"),
    nonce: NONCE,
    authority: await PublicKey.createProgramAddress([
        new PublicKey("ADDRESS_OF_POOL").toBuffer(),
      Uint8Array.from([NONCE])
    ],
    ORCA_TOKEN_SWAP_ID
    ),
    poolTokenMint: new PublicKey("POOL_TOKEN_MINT"),
    poolTokenDecimals: 6,
    feeAccount: new PublicKey("FEE_ACCOUNT"),
    tokenIds: [token_a.mint.toString(), token_b.mint.toString()],
    tokens: {
    [token_a.mint.toString()]: {
      ...token_a,
      addr: new PublicKey("TOKEN_A_VAULT"),
    },
    [token_b.mint.toString()]: {
      ...token_b,
      addr: new PublicKey("TOKEN_B_VAULT"),
    },
    },
    curveType: CurveType.ConstantProduct,
    feeStructure: {
    traderFee: Percentage.fromFraction(TRADER_FEE_NUM, TRADER_FEE_DENOM),
    ownerFee: Percentage.fromFraction(OWNER_FEE_NUM, OWNER_FEE_DENOM),
    },
  });

  const params = TOKENA_TOKENB_params;
  console.log("pool address", params.address.toBase58());
  console.log("nonce", params.nonce);
  console.log("authority", params.authority.toBase58());
  console.log("pool token mint", params.poolTokenMint.toBase58());
  console.log("fee account", params.feeAccount.toBase58());
```

```
  console.log("token ids", params.tokenIds.map((t) => t.toString()));
  console.log("token a name", params.tokens[params.tokenIds[0]].name);
  console.log("token a mint", params.tokens[params.tokenIds[0]].mint.toBase58());
  console.log("token a scale", params.tokens[params.tokenIds[0]].scale);
  console.log("token a vault", params.tokens[params.tokenIds[0]].addr.toBase58());
  console.log("token b name", params.tokens[params.tokenIds[1]].name);
  console.log("token b mint", params.tokens[params.tokenIds[1]].mint.toBase58());
  console.log("token b scale", params.tokens[params.tokenIds[1]].scale);
  console.log("token b vault", params.tokens[params.tokenIds[1]].addr.toBase58());
  console.log("trander fee", params.feeStructure.traderFee.toString());
  console.log("owner fee", params.feeStructure.ownerFee.toString());
}

main();
```

SAMPLE for SAMO/USDC

```
import { PublicKey, Connection } from "@solana/web3.js";
import { ORCA_TOKEN_SWAP_ID, Percentage, Network } from "@orca-so/sdk";
import { OrcaPoolParams, CurveType } from '@orca-so/sdk/dist/model/orca/pool/pool-types';
import { OrcaPoolImpl } from '@orca-so/sdk/dist/model/orca/pool/orca-pool';
import Decimal from 'decimal.js';

async function main() {
  const token_a = {
    tag: "SAMO",
    name: "Samoyed Coin",
    mint: new PublicKey("7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU"),
    scale: 9
  };
  const token_b = {
    tag: "USDC",
    name: "USD Coin",
    mint: new PublicKey("EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"),
    scale: 6
  };

  const samo_usdc_params: OrcaPoolParams = Object.freeze({
    address: new PublicKey("Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow"),
    nonce: 252,
    authority: await PublicKey.createProgramAddress([
        new PublicKey("Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow").toBuffer(),
        Uint8Array.from([252])
      ],
      ORCA_TOKEN_SWAP_ID
    ),
    poolTokenMint: new PublicKey("6VK1ksrmYGMBWUUZfygGF8tHRGpNxQEWv8pfvzQHdyyc"),
    poolTokenDecimals: 6,
    feeAccount: new PublicKey("9U8UF7d8kBvsS25XoZnjmVQ9vGkP4BUnHJgfc615BvG1"),
    tokenIds: [token_a.mint.toString(), token_b.mint.toString()],
    tokens: {
    [token_a.mint.toString()]: {
      ...token_a,
      addr: new PublicKey("7jwHW4Lw3nVaSJXskN5pUoKU6YB9RBVfZtGBp3VbR43U"),
    },
    [token_b.mint.toString()]: {
      ...token_b,
```

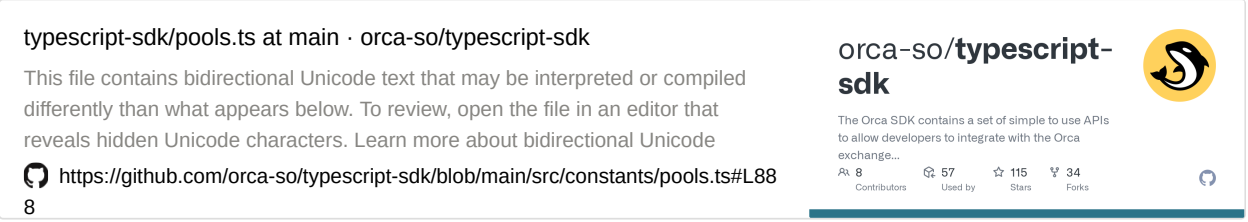
```
      addr: new PublicKey("G7GqjxK9EaJMeFfoFTSy9WfH8uurgQkbNQCREWAc56DZ"),
    },
    },
    curveType: CurveType.ConstantProduct,
    feeStructure: {
    traderFee: Percentage.fromFraction(25, 10000),
    ownerFee: Percentage.fromFraction(5, 10000),
    },
  });

  const params = samo_usdc_params;
  console.log("pool address", params.address.toBase58());
  console.log("nonce", params.nonce);
  console.log("authority", params.authority.toBase58());
  console.log("pool token mint", params.poolTokenMint.toBase58());
  console.log("fee account", params.feeAccount.toBase58());
  console.log("token ids", params.tokenIds.map((t) => t.toString()));
  console.log("token a name", params.tokens[params.tokenIds[0]].name);
  console.log("token a mint", params.tokens[params.tokenIds[0]].mint.toBase58());
  console.log("token a scale", params.tokens[params.tokenIds[0]].scale);
  console.log("token a vault", params.tokens[params.tokenIds[0]].addr.toBase58());
  console.log("token b name", params.tokens[params.tokenIds[1]].name);
  console.log("token b mint", params.tokens[params.tokenIds[1]].mint.toBase58());
  console.log("token b scale", params.tokens[params.tokenIds[1]].scale);
  console.log("token b vault", params.tokens[params.tokenIds[1]].addr.toBase58());
  console.log("trander fee", params.feeStructure.traderFee.toString());
  console.log("owner fee", params.feeStructure.ownerFee.toString());
}

main();
```

OUTPUT of SAMPLE code

> pool address Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow
> nonce 252
> authority AB4rTE2JiKFhnfynUQCovbW75CUxT9LxcJX2SDTbY9gy
> pool token mint 6VK1ksrmYGMBWUUZfygGF8tHRGpNxQEWv8pfvzQHdyyc
> fee account 9U8UF7d8kBvsS25XoZnjmVQ9vGkP4BUnHJgfc615BvG1
> token ids [
> '7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU',
> 'EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v'
>]
> token a name Samoyed Coin
> token a mint 7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU
> token a scale 9
> token a vault 7jwHW4Lw3nVaSJXskN5pUoKU6YB9RBVfZtGBp3VbR43U

> token b name USD Coin
> token b mint EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v
> token b scale 6
> token b vault G7Gqjxk9EaJMeFfoFTSy9WfH8uurgQkbNQCREWAc56DZ
> trander fee 25/10000
> owner fee 5/10000

You can confirm the correctness of this definition with comparing to the definition of SAMO/USDC pool in SDK again.



4. Making OrcaPoolImpl instance

Finally, we can use defined pool!

But orca.getPool() doesn't support pool defined at out of SDK.

So we need to make OrcaPoolImpl instance, which is the return value of orca.getPool(), by ourselves.

Please append the code as follows to the last of main function.

Now you can get the quote of swapping 1 TokenA for TokenB.

```
const connection = new Connection("https://ssc-dao.genesysgo.net/");
const pool = new OrcaPoolImpl(connection, Network.MAINNET, TOKENA_TOKENB_params);

const input_a_amount = new Decimal(1);
const quote = await pool.getQuote(pool.getTokenA(), input_a_amount);
console.log("quote min amount", quote.getMinOutputAmount().toDecimal().toString(), pool.getTokenB().tag);
```

SAMPLE for SAMO/USDC

```
import { PublicKey, Connection } from "@solana/web3.js";
import { ORCA_TOKEN_SWAP_ID, Percentage, Network } from "@orca-so/sdk";
```

```
import { OrcaPoolParams, CurveType } from '@orca-so/sdk/dist/model/orca/pool/pool-types';
import { OrcaPoolImpl } from '@orca-so/sdk/dist/model/orca/pool/orca-pool';
import Decimal from 'decimal.js';

async function main() {
  const token_a = {
    tag: "SAMO",
    name: "Samoyed Coin",
    mint: new PublicKey("7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU"),
    scale: 9
  };
  const token_b = {
    tag: "USDC",
    name: "USD Coin",
    mint: new PublicKey("EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"),
    scale: 6
  };

  const samo_usdc_params: OrcaPoolParams = Object.freeze({
    address: new PublicKey("Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow"),
    nonce: 252,
    authority: await PublicKey.createProgramAddress([
        new PublicKey("Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow").toBuffer(),
        Uint8Array.from([252])
      ],
      ORCA_TOKEN_SWAP_ID
    ),
    poolTokenMint: new PublicKey("6VK1ksrmYGMBWUUZfygGF8tHRGpNxQEWv8pfvzQHdyyc"),
    poolTokenDecimals: 6,
    feeAccount: new PublicKey("9U8UF7d8kBvsS25XoZnjmVQ9vGkP4BUnHJgfc615BvG1"),
    tokenIds: [token_a.mint.toString(), token_b.mint.toString()],
    tokens: {
    [token_a.mint.toString()]: {
      ...token_a,
      addr: new PublicKey("7jwHW4Lw3nVaSJXskN5pUoKU6YB9RBVfZtGBp3VbR43U"),
    },
    [token_b.mint.toString()]: {
      ...token_b,
      addr: new PublicKey("G7Gqjxk9EaJMeFfoFTSy9WfH8uurgQkbNQCREWAc56DZ"),
    },
    },
    curveType: CurveType.ConstantProduct,
    feeStructure: {
    traderFee: Percentage.fromFraction(25, 10000),
    ownerFee: Percentage.fromFraction(5, 10000),
    },
  });

  const params = samo_usdc_params;
  console.log("pool address", params.address.toBase58());
  console.log("nonce", params.nonce);
  console.log("authority", params.authority.toBase58());
  console.log("pool token mint", params.poolTokenMint.toBase58());
  console.log("fee account", params.feeAccount.toBase58());
  console.log("token ids", params.tokenIds.map((t) => t.toString()));
  console.log("token a name", params.tokens[params.tokenIds[0]].name);
  console.log("token a mint", params.tokens[params.tokenIds[0]].mint.toBase58());
  console.log("token a scale", params.tokens[params.tokenIds[0]].scale);
  console.log("token a vault", params.tokens[params.tokenIds[0]].addr.toBase58());
  console.log("token b name", params.tokens[params.tokenIds[1]].name);
  console.log("token b mint", params.tokens[params.tokenIds[1]].mint.toBase58());
  console.log("token b scale", params.tokens[params.tokenIds[1]].scale);
```

```
  console.log("token b vault", params.tokens[params.tokenIds[1]].addr.toBase58());
  console.log("trander fee", params.feeStructure.traderFee.toString());
  console.log("owner fee", params.feeStructure.ownerFee.toString());

  const connection = new Connection("https://ssc-dao.genesysgo.net/");
  const pool = new OrcaPoolImpl(connection, Network.MAINNET, samo_usdc_params);

  const input_a_amount = new Decimal(1);
  const quote = await pool.getQuote(pool.getTokenA(), input_a_amount);
  console.log("quote min amount", quote.getMinOutputAmount().toDecimal().toString(), pool.getTokenB().tag);
}

main();
```

OUTPUT for SAMPLE code

> pool address Epvp7qMYAF21VVjacdB3VfKn6nnXQSF4rGYu8sD6Bkow
> nonce 252
> authority AB4rTE2JiKFhnfynUQCovbW75CUxT9LxcJX2SDTbY9gy
> pool token mint 6VK1ksrmYGMBWUUZfygGF8tHRGpNxQEWv8pfvzQHdyyc
> fee account 9U8UF7d8kBvsS25XoZnjmVQ9vGkP4BUnHJgfc615BvG1
> token ids [
> '7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU',
> 'EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v'
>]
> token a name Samoyed Coin
> token a mint 7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU
> token a scale 9
> token a vault 7jwHW4Lw3nVaSJXskN5pUoKU6YB9RBVfZtGBp3VbR43U
> token b name USD Coin
> token b mint EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v
> token b scale 6
> token b vault G7Gqjxk9EaJMeFfoFTSy9WfH8uurgQkbNQCREWAc56DZ
> trander fee 25/10000
> owner fee 5/10000

> quote min amount 0.005183 USDC

END 👍

BTW, when SAMO to the moon ? 🤣