Token Swap Program

A Uniswap-like exchange for the Token program on the Solana blockchain, implementing multiple automated market maker (AMM) curves.

Audit

The repositoryREADME contains information about program audits.

Available Deployments

Network Version Program Address Testnet 3.0.0 SwapsVeCiPHMUAtzQWZw7RjsKjgCjhwU55QGu4U1Szw Devnet 3.0.0 SwapsVeCiPHMUAtzQWZw7RjsKjgCjhwU55QGu4U1Szw While third-party deployments of token-swap exist on Mainnet Beta, the team has no plans to deploy it themselves.

Check out theprogram repository for more developer information.

Overview

The Token Swap Program allows simple trading of token pairs without a centralized limit order book. The program uses a mathematical formula called "curve" to calculate the price of all trades. Curves aim to mimic normal market dynamics: for example, as traders buy a lot of one token type, the value of the other token type goes up.

Depositors in the pool provide liquidity for the token pair. That liquidity enables trade execution at spot price. In exchange for their liquidity, depositors receive pool tokens, representing their fractional ownership in the pool. During each trade, a program withholds a portion of the input token as a fee. That fee increases the value of pool tokens by being stored in the pool.

This program was heavily inspired byUniswap andBalancer . More information is available in their excellent documentation and whitepapers.

Background

Solana's programming model and the definitions of the Solana terms used in this document are available at:

- https://docs.solana.com/apps
- https://docs.solana.com/terminology

Source

The Token Swap Program's source is available ongithub .

Interface

JavaScript bindings are available that support loading the Token Swap Program on to a chain and issuing instructions.

Example user interface is availablehere .

Operational overview

The following explains the instructions available in the Token Swap Program. Note that each instruction has a simple code example that can be found in theend-to-end tests .

Creating a new token swap pool

The creation of a pool showcases the account, instruction, and authorization models on Solana, which can be very different compared to other blockchains.

Initialization of a pool between two token types, which we'll call "A" and "B" for simplicity, requires the following accounts:

- empty pool state account
- pool authority
- token A account
- token B account
- pool token mint

- pool token fee account
- pool token recipient account
- token program

The pool state account simply needs to be created usingsystem_instruction::create_account with the correct size and enough lamports to be rent-free.

The pool authority is aprogram derived address that can "sign" instructions towards other programs. This is required for the Token Swap Program to mint pool tokens and transfer tokens from its token A and B accounts.

The token A / B accounts, pool token mint, and pool token accounts must all be created (usingsystem_instruction::create_account) and initialized (usingspl_token::instruction::initialize_mint orspl_token::instruction::initialize_account). The token A and B accounts must be funded with tokens, and their owner set to the swap authority, and the mint must also be owned by the swap authority.

Once all of these accounts are created, the Token Swapinitialize instruction will properly set everything up and allow for immediate trading. Note that the pool state account is not required to be a signer oninitialize , so it's important to perform theinitialize instruction in the same transaction as itssystem_instruction::create_account .

Swapping

Once a pool is created, users can immediately begin trading on it using theswap instruction. The swap instruction transfers tokens from a user's source account into the swap's source token account, and then transfers tokens from its destination token account into the user's destination token account.

Since Solana programs require all accounts to be declared in the instruction, users need to gather all account information from the pool state account: the token A and B accounts, pool token mint, and fee account.

Additionally, the user must allow for tokens to be transferred from their source token account. The best practice is tospl_token::instruction::approve a precise amount to a new throwaway Keypair, and then have that new Keypair sign the swap transaction. This limits the amount of tokens that can be taken from the user's account by the program.

Depositing liquidity

To allow any trading, the pool needs liquidity provided from the outside. Using thedeposit_all_token_types ordeposit_single_token_type_exact_amount_in instructions, anyone can provide liquidity for others to trade, and in exchange, depositors receive a pool token representing fractional ownership of all A and B tokens in the pool.

Additionally, the user will need to approve a delegate to transfer tokens from their A and B token accounts. This limits the amount of tokens that can be taken from the user's account by the program.

Withdrawing liquidity

At any time, pool token holders may redeem their pool tokens in exchange for tokens A and B, returned at the current "fair" rate as determined by the curve. In thewithdraw_all_token_types andwithdraw_single_token_type_exact_amount_out instructions, pool tokens are burned, and tokens A and B are transferred into the user's accounts.

Additionally, the user will need to approve a delegate to transfer tokens from their pool token account. This limits the amount of tokens that can be taken from the user's account by the program.

Curves

The Token Swap Program is completely customizable for any possible trading curve that implements theCurveCalculator trait. If you would like to implement a new automated market maker, it may be as easy as forking the Token Swap Program and implementing a new curve. The following curves are all provided out of the box for reference.

Constant product

Theconstant product curve is the well-known Uniswap and Balancer style curve that preserves an invariant on all swaps, expressed as the product of the quantity of token A and token B in the swap.

A_total * B_total = invariant If a trader wishes to put in token A for some amount of token B, the calculation for token B becomes:

(A_total + A_in) * (B_total - B_out) = invariant For example, if the swap has 100 token A and 5,000 token B, and a trader wishes to put in 10 token A, we can solve for theinvariant and thenB_out :

A_total * B_total = 100 * 5,000 = 500,000 = invariant And

(A_total + A_in) * (B_total - B_out) = invariant (100 + 10) * (5,000 - B_out) = 500,000 5,000 - B_out = 500,000 / 110 5,000 - (500,000 / 110) = B_out B_out = 454.5454... More information can be found on theUniswap whitepaper and theBalancer whitepaper .

Constant price

Theconstant price curve is a simple curve that always maintains the price of token A with respect to token B. At initialization, the swap creator sets the cost for 1 token B in terms of token A. For example, if the price is set to 17, 17 token A will always be required to receive 1 token B, and 1 token B will always be required to receive 17 token A.

Note that this curve does not follow traditional market dynamics, since the price is always the same.

Constant price curves are most useful for fixed offerings of new tokens that explicitly should not have market dynamics. For example, a decentralized game creator wants to sell new "SOLGAME" tokens to be used in their game, so they create a constant price swap of 2 USDC per SOLGAME, and supply all of the SOLGAME tokens at swap creation. Users can go to the swap and purchase all of the tokens they want and not worry about the market making SOLGAME tokens too expensive.

Stable (under construction)

Thestable curve fromcurve.fi , has a different shape to prioritize "stable" trading, meaning prices that stay constant through trading. Most importantly, prices don't change as quickly as the constant product curve, so a stable swap between two coins that represent the same value should be as close to 1:1 as possible. For example, stablecoins that represent a value in USD (USDC, TUSD, USDT, DAI), should not have big price discrepancies due to the amount of tokens in the swap.

The curve mirrors the dynamics of the curve More information can be found on theirwhitepaper .

The Token Swap Program implementation of the stable curve is under construction, and a more complete version can be found at thestable-swap-program .

Offset

Theoffset curve can be seen as a combination of the constant price and constant product curve. It follows the constant product curve dynamics, but allows for the pool creator to set an "offset" on one side. The invariant for the curve is:

(A_total) * (B_total + B_offset) = invariant This is useful for initial token offerings, where the token creator wants to sell some new token as a swap without putting up the capital to fund the other side of the swap. This is similar to the constant price curve, but the key difference is that the offset curve captures normal market dynamics, in that the offered token price will increase as it is bought.

For example, a decentralized betting application creator wants to sell new "SOLBET" tokens on the market in exchange for USDC, and they believe each token is worth at least 4 USDC. They create a pool between SOLBET and USDC, funding one side with 1,000 SOLBET, and the other side with 0 USDC, but an offset of 4,000 USDC.

If a trader tries to buy SOLBET with 40 USDC, the invariant is calculated with the offset:

(SOLBET_total) * (USDC_total + USDC_offset) = invariant 1,000 * (0 + 4,000) = 4,000,000

(SOLBET_total - SOLBET_out) * (USDC_total + USDC_offset + USDC_in) = invariant SOLBET_out = 9.901 The trader received 9.901 SOLBET for 40 USDC, so the price per SOLBET was roughly 4.04, slightly higher than the minimum of 4 USDC per SOLBET.

Conversely, if a trader tries to buy USDC with SOLBET immediately after creation, it will fail because there is no USDC actually present in the pool.

Testing

The token-swap program is tested using various strategies, including unit tests, integration tests, property tests, and fuzzing. Since unit tests and integration tests are well-known, we highlight property tests and fuzzing here.

Property testing

Using theproptest crate, we test specific mathematical properties of curves, specifically to avoid leaking value on any trades, deposits, or withdrawals. It is out of scope of this document to explain property testing, but the specific property tests for the Token Swap Program can be found in thecurves andmath portions of the repo.

Fuzzing

Usinghonggfuzz , we regularly test all possible inputs to the Token Swap Program, ensuring that the program does not

crash unexpectedly or leak tokens. It is out of scope of this document to explain fuzzing, but the specific implementation for the program can be found in theinstruction fuzz tests of the repo. Edit this page Previous Presentation Next Token-Lending Program * Audit * Available Deployments * Overview * Background * Source * Interface * Operational overview * * Creating a new token swap pool * * Swapping * * Depositing liquidity * * Withdrawing liquidity * Curves * * Constant product * * Constant price * * Stable (under construction) * * Offset * Testing * * Property testing * * Fuzzing