Swap

Let's Swap 0.1 USDC => ETH

Since this is the first section, I'm going to write it out in full, and from add liquidity on, I'm going to keep it simple and easy to understand.

**Always referExample Codes to see what was the exact implementation of helper functions like toToken, toPoolId, toTokenInfo.

```

Copy

usingTokenLibforToken; /** if you don't want to import Token lib, you could simply use this snippet instead for Token type conversion.

For ERC20, it's just a bytes32 left padded with 0. For Eth(Native token), use 0xEEEE..E as token address.

type Token is bytes32; function toToken(IERC20 tok) pure returns (Token) { return Token.wrap(bytes32(uint256(uint160(address(tok))))); } */ IVault vault=IVault(0x1d0188c4B276A09366D05d6Be06aF61a73bC7535);

uint8constantSWAP=0; uint8constantGAUGE=1;

uint8constantEXACTLY=0; uint8constantAT_MOST=1; uint8constantALL=2;

functionswapExample()external{ addressusdc=0x176211869cA2b568f2A7D4EE941E073a821EE1ff; addressvc=0xcc22F6AA610D1b2a0e89EF228079cB3e1831b1D1; addresseth=address(0); addressusdc_eth_pool=vault.getPair(usdc,eth); addressusdc_eth_lp=usdc_eth_pool;

IERC20(usdc).approve(address(vault),type(uint256).max); // you can optimize gas by batching operations. // this example will execute them separately for clarity

//swap usdc->eth / Remember that Execute takes 3 params : Token[], deposit[], Op[]. Let's construct them one by one. First, deposit[] is rarely used, so you could just set it as a zero array with the same length as Token array. /

//Step 1. Prepare Token array //2 tokens, USDC & ETH are involved in this swap. Token[]memorytokens=newToken

//Token constant NATIVE_TOKEN = Token.wrap(0xEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE); //ref: lib/Token.sol tokens[0]=toToken(IERC20(usdc)); tokens[1]=eth==address(0)?NATIVE_TOKEN:toToken(IERC20(eth));// =NATIVE_TOKEN for eth

//Step 2. Prepare Op array /* struct VelocoreOperation { // 1byte optype(SWAP/STAKE/CONVERT/VOTE/handleUserBalance) + (11bytes 0 padding) + 20bytes related pool address bytes32 poolId;

// 1byte index at Token[] above + 1 byte amount Type(exactly/at most/vitual balance) // + (14bytes padding)+ 16bytes int128 desiredAmount(=minimum Amt) bytes32[] tokenInformations;

//usually empty. "" bytes data; } */

// We are doing only one operation here. Swap. So op length is 1. VelocoreOperation[]memoryops=newVelocoreOperation
// optype==SWAP, pool=usdc_eth_pool ops[0].poolId=toPoolId(SWAP,usdc_eth_pool);

ops[0].tokenInformations=newbytes32; // Index for usdc is 0 at Token[], Exactly 0.1e6 USDC is used. pool receives USDC so amount is positive. ops[0].tokenInformations[0]=toTokenInfo(0x00,EXACTLY,0.1e6); // Index for eth is 1 at Token[]. We didn't set slippage here, but you could calculate the minimum amount and use the number instead of 0. // ETH is expected to flow out of the pool, so the sign of int128 should be negative. if the minimum amount you want is 0.1 eth, use int128(-0.1). // That's why it's called AT_MOST, not AT_LEAST. ops[0].tokenInformations[1]=toTokenInfo(0x01,AT_MOST,0); ops[0].data="";

//Step 3. execute() !! returnexecute(tokens,newint128,ops); }

```

Last updated4 months ago On this page