How to interact with VELOCORE

Execute(), one function to do all the SWAP/LPing/Vote/Stake/bribe/etc. As you can see in the architecture above, all actions in Velocore are done through the Vault contract. You don't have to go to each address of Router, Factory, Gauge, etc. and interact with them separately; all you need is the Vault contract address to do everything.

Vault.execute()

Vault.execute() is the main function for transactions with Velocore V2. This function receivesVelocoreOperation from the user, executes them in sequence while tracking the user's virtual balance, and finally carries out token transfers.

Tokens are transferred only after all operations are executed, and the virtual balance can even go negative, which indicates an impending payment by the user.

```

Copy functionexecute( Token[] tokenRef, int128[] deposit, VelocoreOperation[] ops )

```

1. tokenRef

tokenRef is an array ofToken(type bytes32) (less than 256 elements), listing all tokens involved in transactions. For instance, [USDC, ETH, VC].

Velocore V2 supports not only ERC20 but also ERC721, ERC1155, and native tokens. Tokens are represented as a wrapped bytes32 (Token), specifically:

- 1 byte of token type
-
  - 0x00
-
  - : ERC20
-
  - 0x01
-
  - : erc721
-
  - 0x02
-
  - : erc1155
- *
- 11 bytes of token id:
-
  - always 0 for ERC20
-
  - NFT id or Token id for ERC721 and erc1155
-
  - token ids greater than 2^88 - 1 is unsupported
- *
- 20 bytes of token address
- native token is represented as 0xeeeee...eeeee.
-

1. deposit

deposit is an array of positiveint128 matching the length of tokenRef. Vault withdraws specified amounts from the user before transactions and credits these to thevirtual balances . It's useful for selling tax-on-transfer tokens.Usually, this is an array of zeroes .

1. VelocoreOperation

VelocoreOperation is an abstraction of the swap process. Users send VelocoreOperations to the Vault to perform swaps. Each operation includes operation type (e.g., swap, stake, vote), the pool to interact with, and details likeToken , desired amount, and nature of the amount.

An example of a transaction that pays USDC to buy ETH is conceptualized as:

```
```

Copy type: swap pool: USDC-ETH pool details: [(USDC, "exactly +1000000"), (ETH, "at most -1000")]

```

```

Copy structVelocoreOperation{ bytes32poolId; bytes32[] tokenInformations; bytesdata; }

```

We use a compressed encoding to optimize calldata usage.

- poolId
- : A bytes32 combination of
-
  - 1 byte ofoperation type
-
  - :
-
  -
    - 0x00: swap
-
  -
    - 0x01: stake
-
  -
    - 0x02: convert
-
  -
    - 0x03: vote
-
  -
    - 0x04: userBalance operations

---

-
  - (11 bytes of unused bytes)
-
  - 20 bytes ofPool
-
  - address
- *
- tokenInformation
- : A bytes32 combination of:
-
  - 1 byte (uint8) of the index of theToken
-
  - intokenRef
-
  - 1 byte ofamountType
-
  -
    - 0x00: exactly
-
  -
    - 0x01: at most
-
  -
    - 0x02: equal to the virtual balance

---

-
  - 14 bytes of unused bytes
-
  - 16 bytes ofint128
-
  - , thedesiredAmount
-
  -
    - the transaction will fail, regardless of theamountType
```

-
    -
        - , if the pool returns less than this.

- *
- data
- : Auxiliary data, typically empty bytes.
-

tokenInformation must contain for all tokens involved, including VC emitted during (un)staking and veVC deposited/withdrawn during voting.

Pool implementations define specific requirements for operationType, tokenInformation, and data. The Vault only ensures the result is less than desiredAmount. Each VelocoreOperation can be viewed as a desired token balance change vector, with auxiliary data.

Beware of malicious pools that might confiscate deposited tokens. Users shouldn't use Pool addresses blindly.

Operation Type Differences

Swap applies to any token exchangesnot involving VC emission or voting . It includes swap, veVC conversion, and LP deposit/withdrawal.

- ISwap
- refers to pools supporting this opertaion
- ForCPMM(Volatile pools)
- andWombat pools
- , LP deposit/withdrawal can be performed by 'buying/selling' LP tokens from/to underlying tokens.
-
    - This is an important concept. Adding/removing liquidity is not a separate function, it's the act of swapping tokens for LP tokens!
- *
- VC
- andveVC
- are ISwap; VC can be 'bought' with old VC; and veVC can be bought with old veNFTs or new VC. (This was only used in zkSync in the past, migrating from v1 to v2, and is now obsolete)
- the plan is to make veVC act as the liquidity pool for veVC; this is not implemented yet.
-

Stake diverges fromswap as the Vault calculates and emits VC before operation. It is used for interacting with gauges.

- IGauge
- refers to pools supporting this opertaion
- CPMM
- is bothISwap
- andIGauge
- .
- Wombat
- pools have multiple gauges; one for each lp token.
- Harvesting can usually be performed by specifying [VC, at most, 0], without any lp tokens.
-

Convert involves actual token transfer, different from swap. The Vault transfersdesiredAmount to the pool before calling the pool, and the pool sends the output to the vault.

- IConverter
- refers to pools supporting this opertaion.
- Vault only monitors specified tokens balance changes; any unspecified tokens received will be lost.
- This is useful for converting tokens to/from wrapped tokens (e.g. WETH or rfUSDC) or for flash loans.
-

Vote is significantly different from swap. This operation requires the pool to beIGauge , andtokenInformation must include veVC and any bribe tokens user wants to receive. Like stake, it sends VC emission and calculates and sends any bribes attached to the gauge. To (un)vote, users must specify thedesiredAmount of the veVC. Any bribes not included intokenInformation will be credited touserBalance instead.

- Harvesting can usually be performed by specifying [veVC, exactly, 0], along with bribe tokens.
-

userBalance :

userbalance is the user's token-specific balance held in the vault, unused, and available for withdrawal at any time, stored in userbalance[useraddress][Token]. pro traders can save gas by preloading their tokens into userBalance to avoid having to perform a token transfer every time they trade.

Pool in this operation can be any address, allowing depositing any tokens to any address. Withdrawal is only possible from your userBalance.

Vault.query()

```

Copy functionquery( addressuser, Token[]memorytokenRef, int128[]memorydeposit, VelocoreOperation[]calldataops )publicreturns(int128[]memory)

```

To determine the expected amounts received from transactions, useVault.query() . The function parameters are the same as before, with the addition of user, which specifies the transaction maker. The function returns an array of int128 representing the final transactional changes in token balances.

Pseudocode Example

```

Copy // helper functions

consttoToken=(spec:string,id:BigNumber,addr:string)=> solidityPack( ["uint8","uint120","address"], [["erc20","erc721","erc1155"].indexOf(spec),id,addr] ) constpoolId=(poolAddress:string)=> solidityPack( ["bytes1","uint120","address"], ["0x00",0,poolAddress] )

consttokenInformation=(index:number,amountType:string,amount:BigNumber)=> solidityPack(["uint8","uint8","uint112","int128"],[ index, ["exactly","at most","all"].indexOf(amountType), 0, amount ] )

constcompileAndExecute=(ops)=>{ consttokenRef=[...newSet(ops.map(x=>x[1].map(i=>i[0])))].sort(); returnvault.execute( tokenRef, (newArray(tokenRef.length)).fill(0), ops.map(op=>({ poolId:op[0] tokenInforamtions: op[1].map(i=>tokenInformation(tokenRef.indexOf(i[0]),i[1],i[2])).sort(), data:[] })) ); }

constusdc=toToken("erc20",0,"0xUSDC"); constusdt=toToken("erc20",0,"0xUSDT"); consteth="0xEEE...EEE";

// actually make transactions awaitcompileAndExecute([ [poolId("0x12311..."),[ [usdc,"exactly",1234], [usdt,"at most",0] ]], [poolId("0x1ab3234..."),[ [usdt,"all",INT128_MAX], [eth,"at most",-1234], ]] ])

```

Last updated4 months ago On this page