Schemas

All the instances and solutions of the batch auction problem are formatted inJSON . In this section, we describe these schemas.

note More details can be foundhere . caution To avoid precision loss, some numerical literals are encoded as strings, referred below asstringified .

Instances (input)

The instance json that solver engines currently receive contains six keys:

- id
- tokens
- orders
- liquidity
- effectiveGasPrice
- deadline

We now explain what each of these entries contains.

id

This key is an (internal) integer identifier for the auction, that is encoded and sent as a string. In the case where the instance corresponds to a quote request, this identifier is set tonull .

tokens

This key lists all tokens that appear in some order or AMM in the batch auction instance. It is a dictionary, mapping the token key (which is the smart contract address of the token) to the following information:

- decimals
- : an integer equal to the number of decimals of the token.
- symbol
- : a string denoting the shorthand name of the token (e.g., "WETH", "DAI").
- referencePrice
- : a float that corresponds to the price of the smallest denomination of the token with respect to areference token
- (for mainnet, the reference token is WETH, and its referencePrice is 1000000000000000000). Only tokens that are traded by at least a user order will necessarily have non-null referencePrice, while the rest of the tokens are allowed to have anull
- referencePrice. These prices are used when evaluating the quality of a given solution, and can be thought of as a way of converting and expressing all relevant quantities in WETH (note that, initially, the surplus of different orders can be denominated in different tokens), and aggregating them all in a single value, denominated in WETH.
- availableBalance
- : a stringified integer that describes the amount (in the token's lowest denomination) of the token currently stored in the settlement contract (internal buffers
-). This information is relevant when a solver attempts tointernalize an interaction
- .
- trusted
- : this is a boolean flag that specifies whether the contract is willing to store the token as part of aninternalized interaction
- .

We now share two example token entries corresponding toWETH andUSDC :

"0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2" :

{ "decimals" :

18 , "symbol" :

"WETH" , "referencePrice" :

"1000000000000000000" , "availableBalance" :

"590308372204674634" , "trusted" :

true } "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48" :

{ "decimals" :

6 , "symbol" :

"USDC" , "referencePrice" :

"449666048539228625975640064" , "availableBalance" :

"2625685411" , "trusted" :

true } We clarify a few things regarding the entries above. The referencePrice of a token, viewed in isolation, is just a number and does not have inherent meaning. However, the referencePrice suggests an (approximate) exchange rate between two tokens, and this is the reason these prices are used to convert a quantity denominated in some token to WETH. Specifically, we usually have thatWETH has a referencePrice of 1000000000000000000, which should be interpreted as 1 wei = 1/1018 WETH (sinceWETH has 18decimals) has a price of 1000000000000000000. This implies that thereferencePrice ofUSDC , which here is set to 449666048539228625975640064, means that one atom of USDC, i.e., 1/106 of 1USDC has a price of 449666048539228625975640064, given that 1 wei has price 1000000000000000000. This suggests that 1 USDC is equal to referencePrice(USDC) * 106 / (referencePrice(WETH) * 1018) = 0.0004496660485392286 WETH, or equivalently, that 1 WETH is equal to 2223.87259 USDC.

note Both tokens above are accepted for internalization.

orders

This key maps to a list containing the set of orders in the batch. Each entry in the dictionary corresponds to an order, which is a dictionary containing the following:

- uid
- : this is the unique identifier of the order.
- sellToken
- : a string denoting the address of the sell token.
- buyToken
- : a string denoting the address of the buy token.
- sellAmount
- : a stringified integer denoting the limit amount that is being sold, measured in terms of the smallest denomination of the sell token.
- buyAmount
- : a stringified integer denoting the limit amount that is being bought. Similar to thesellAmount
- , it is measured in terms of the smallest denomination of the buy token.
- feeAmount
- : a stringified integer denoting the signed fee attached to the order, which is always denominated in thesellToken
- .
- kind
- : a string of the set {"sell", "buy"}, describing whether the order is asell
- orbuy
- order.
- partiallyFillable
- : a boolean indicating whether the order may be partially matched (true
-), or if it is Fill-or-Kill order (false
-).
- class
- : a string of the set {"market", "limit", "liquidity"}, indicating the order class.
- We clarify here that allmarket
- andliquidity
- orders have a potentially non-zero predetermined fee, while alllimit
- orders have necessarily a zero signed fee, and the actual fee charged to the order is computed and provided by the solvers when they propose an execution of such an order. More details are provided in thesolutions section
- .

An example Fill-or-Kill user limit buy order that sells 1000COW for at least 284.138335 USDCUSDC is given below:

{ "uid" :

"0xaa4eb7b4da14b93ce42963ac4085fd8eee4a04170b36454f9f8b91b91f69705387a04752e516548b0d5d4df97384c0b22b64917965a801c1" , "sellToken" :

"0xdef1ca1fb7fbcdc777520aa7f396b4e015f497ab" , "buyToken" :

"0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48" , "sellAmount" :

"10000000000000000000" , "buyAmount" :

"284138335" , "feeAmount" :

"0" , "kind" :

"sell" , "partiallyFillable" :

false , "class" :

"limit" } The above entry should be interpreted as follows. It is a Fill-or-Kill order since the flagpartiallyFillable is set tofalse . Moreover, it is a sell order since itskind is set tosell . Finally, this is alimit order, meaning that it has a zero-signed fee, which implies that the solver is free to choose an appropriate fee to cover its execution cost. This means that, if executed, the user will send a total of 10000000000000000000 COW atoms to the settlement contract and, no matter how much fee the solver will charge, the user is guaranteed to receive at least 284138335 USDC atoms.

liquidity

This key is a list of all the publicly available AMMs/limit orders that are made available for use by the default implementation of the Driver, available to the solver engines that request such liquidity. Internally, this is also known asbaseline liquidity. Each entry describes the current state of an AMM or an existing public limit order. Currently, there are 5 different types of liquidity provided by the Driver:

- constant product pools, i.e., Uniswap v2 type pools on 2 tokens.
- weighted Product pools, i.e., Balancer type weighted product pools on N tokens.
- stable pools, i.e., Curve type stable pools on N tokens.
- concentrated liquidity pools, i.e., Uniswap v3 type concentrated liquidity pools on 2 tokens.
- foreign limit orders, i.e., external 0x type limit orders on 2 tokens.

More information about the exact descriptions of these pools can be foundhere .

effectiveGasPrice

This key is a single entry that is a stringified integer describing an estimate of the effective gas price for the winning solution, denominated in wei.

deadline

This key is a single entry that is a string corresponding to a time stamp. This timestamp specifies the deadline by which a solution to the auction is required. Responses that go beyond this deadline are considered invalid.

Solutions (output)

The response of a solver engine is a dictionary that contains all the information that is needed to execute the proposed solution. There is a single key, "solutions", that maps to a list of proposed solutions; this means that a solver can propose multiple solutions. The simplest possible response is theempty solution , which corresponds to the case where no orders are executed, and is described by the empty list as follows:

{ "solutions" : [ ] } Each entry of thesolutions list describes a solution, which is a dictionary that consists of the following.

id

This key maps to a non-negative integer that can be thought of as an index of the proposed solution and must be unique for each proposed solution. E.g., if a solver decides to propose two solutions, they can use index 0 and 1.

prices

This key specifies the uniform clearing price vector (UCP) that governs the proposed solution. It is a dictionary mapping each token traded by a user order to its computed price. Each price is an stringified unsigned integer, and the whole vector is scale invariant, meaning that multiplying each entry by the same constant will not change anything; thus the scaling is chosen by the solver. We also stress that a solution need only contain prices for the tokens appearing in the executed user orders, and that solvers are free to choose the unit of measure they will use.

An example containing the computed prices ofUSDC ,COW andWETH is given below.

"prices" :

{ "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48" :

"389193245742363509576247472" , "0xDEf1CA1fb7FBcDC777520aa7f396b4E015F497aB" :

"137298311435590" , "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2" :

"1000000000000000000" } The above entries should be interpreted as follows:

- 1 wei (3rd entry) has a price of 1000000000000000000 = 1018
- .
- The lowest denomination ofUSDC
- (1st entry), i.e., 1 / 106
- ofUSDC
- , has a price of 389193245742363509576247472 relative to the price of 1018
- that wei has. This translates, for example, to 1WETH
- having a price of1000000000000000000
- .
- 1
- 0
- 18

- 3891932457423635095762474 72
- .
- 1
- 0
- 6
- ≈
- 2569.417
- \frac{1000000000000000000 \cdot 10^{18}}{3891932457423635095762474 72 \cdot 10^{6}} \approx 2569.417
- 3891932457423635095762474 72
- .
- 1
- 0
- 6
- 1000000000000000000
- .
- 1
- 0
- 18
-
- ≈
- 2569.417
- USDC
- .
- The lowest denomination ofCOW
- (2nd entry) is 1 / 1018
- , and it has a price of 137298311435590. This translates, for example, to 1COW
- having a price of137298311435590
- .
- 1
- 0
- 18
- 3891932457423635095762474 72
- .
- 1
- 0
- 6
- ≈
- 0.35277
- \frac{137298311435590 \cdot 10^{18}}{3891932457423635095762474 72 \cdot 10^{6}} \approx 0.35277
- 3891932457423635095762474 72
- .
- 1
- 0
- 6
- 137298311435590
- .
- 1
- 0
- 18
-
- ≈
- 0.35277
- USDC.

trades

This key maps to a list of all orders that were selected for execution. Each trade is a dictionary that contains the following entries:

- kind
- : this is string of the set {"fulfillment", "jit"}, which corresponds to an order existing in the orderbook, or a just-in-time liquidity order placed by the solver, respectively.
- order
- : in case of a "fulfillment" trade, theuid
- of the order is provided here as a string. In case it is a just-in-time liquidity order, the specifications of the order are explicitly given as a dictionary; more details can be foundhere
- .
- fee
- : this entry exists only for "fullilment" trades, and maps to a stringified integer describing the fee of the order (either pre-signed or solver computed), denominated in the sell token.
- executedAmount
- : this is a stringified integer corresponding to the sell amount (for sell orders) or the buy amount (for buy orders) that would get executed. Note that this is amount is "separate" from the "fee" amount that was described above; this, for example, means that for a sell "fulfillment" order, the trader will send a total offee + executedAmount
- sell tokens to the contract. We also stress that it is theexecutedAmount
- where uniform clearing prices are being applied to.

interactions

This key maps to a list of all interactions that are executed in the proposed solution, in the order they are provided in the list. Every interaction is either an AMM/liquidity order included in the input json sent to solvers, or a custom interaction the solver computed. Each interaction is a dictionary that contains the following entries:

- kind
- : this is a string of the set {"liquidity", "custom"}, that specifies whether the interaction refers to an AMM/liquidity order included in the input json sent to the solvers, or it is a custom interaction.
- internalize
- : this is a boolean flag that specifies whether the interaction can be internalized or not. More details about internalizations can be found in the section right below.
- Depending on the "kind", the rest of the keys are different, and the details and specifications of those can be found here
- .

Using internal buffers

We now discuss some additional functionality that solvers are allowed to use. Since the settlement contract holds balances of multiple tokens, solvers are in certain cases allowed to "internalize" an interaction, in order to save on gas. More precisely, if there is an AMM interaction that buys token A and sells token B, a solver can decide to internalize the interaction if and only if the following two conditions are satisfied:

1. Token A is a trusted token, i.e., the correspondingtrusted
2. flag is set totrue
3. . This means that the protocol is happy to store this token in the settlement contract.
4. There is enough balance of sell token B, i.e., at least as much amount as the sell amount of the AMM interaction. This is revealed by theavailableBalance
5. entry in the token description.

If both conditions are satisfied, a solver can set theinternalize flag totrue in order to internalize the interaction:

"internalize" :

true In such a case, the default driver will remove the interaction, and so the solution will end up using less gas, get better ranking, and also be risk-free (at least the part involving the internalized AMM interaction).

score

The score is a key that describes the "bid" a solver makes for the batch in thesolver auction , as it will get ranked according to it. The protocol picks the solution with the highest score, given that it is strictly positive. The score maps to a dictionary with two entries. The first entry is the following:

- kind
- : this is a string of the set {"solver", "riskAdjusted"}, that determines whether a solver will explicitly provide a score or will delegate the score computation to the default driver by only specifying a probability of success for the proposed solution.

If we have"kind": "solver" , then there is a second entry, labeledscore , that corresponds to a stringified float that specifies the score attached to the solution. On the other hand, if we have"kind": "riskAdjusted" , then there is a second entry, labeledsuccessProbability , that is a stringified float that specifies the success probability of the proposed solution.

note All solvers are encouraged to submit a score via"kind": "riskAdjusted" andsuccessProbability , as it has been observed to be more accurate and fully protects solvers from overbidding (which is prohibited by social consensus rules; seehere for more information). Edit this page Previous Slippage accounting Next Intents