

Order book

The orderbook is the main API that traders, UIs and other integrations use to interact with CoW Protocol.

Its implementation can be found [here](#) . The API is documented in detail [here](#) .

Overview

By and large the orderbook is a [CRUD](#) service for the state of the protocol. It connects to the database which is shared with the [autopilot](#) to insert or update orders and serves information about existing orders as well as previous trades and auctions.

Architecture

The standard trader interaction flow can be seen in the following sequence diagram:

After selecting the parameters of their trade, most traders want to see an estimate of how much tokens they will receive in order to pick a reasonable limit price. For that they send a quote request to the orderbook. The orderbook is connected to a subset of solvers that are participating in the competitions (technically their [drivers](#)) and creates price estimates by sending these solvers a request to solve a batch auction containing just a single order.

The orderbook may simulate quotes if possible to make sure their proposed solution would pass given the current state of the chain and picks the one maximizing the traders' output (the solver winning the quote is rewarded by the protocol). The quote is turned into an order that is returned to the trader and is ready to be signed by the user.

Once signed, the trader will post the order for inclusion in the next batch. At this stage the orderbook will verify the order is valid and either fetch the existing quote or create a new one (the "expected" out amount at order placement time is relevant for order classification and fee policies). Finally it stores the order in the database, so that the autopilot can index it, and returns the unique identifier to the trader.

Now it's the protocol's and solvers' turn to distribute and match the order. During this time, the trader may request status updates about their order from the orderbook and finally request information about the executed trade and solver competition.

Methodology

The orderbook exposes a variety of different endpoints. The majority of them are simple [REST](#) interfaces on top of the data model that is stored in the database. However, there are some more involved operations, which are worth explaining in more detail:

Getting quotes

The quoting process piggy-backs on the solver competition. Solver are tasked in finding a matching for an auction instance including only one order (the order to be quoted). These quote auctions have a significantly shorter deadline to solve "quote" auctions as they are fundamentally simpler to solve.

Quotes can be requested using either the fast or optimal quality type.

- Optimal quotes
 - : wait for all connected solvers to either return a response or time out before yielding a result
- Fast quotes
 - : return the best estimate as soon as a threshold of solvers (i.e. two) have returned a successful estimate

The API also exposes another price-related route, `/native_price` , which can be used to get an approximate estimate of the dollar (or ETH) value for a given amount of tokens. These price estimates use fixed amounts and are more heavily cached allowing for faster response times.

Placing orders

Before an order is accepted in the database it is validated. Validation steps include

- Asserting the order is well formed and all fields are properly encoded
- The user has sufficient balance/allowance to place this order
- The [signature type](#)
 - is supported and the signature is valid
- The pre-image for the order's [appData](#)
- hash is either provided in the request or publicly available on IPFS (to ensure any additional information about the order can be properly interpreted by the orderbook)

Each order is also associated with a quote for classification as either in or out of market price. This may have an effect on the fee policy used. For this either the trader provided quote id is looked up or a fresh quote is created.

Considerations

The orderbook synchronizes state with the autopilot via a shared database. It communicates with solvers to produce price estimates. It uses IPFS to update and fetch [appData](#) documents. It also depends on on-chain state for:

- checking the trader has sufficient balance/approval to place the order
- simulate smart contract signatures and hooks [Edit this page](#) [Previous Arbitrating auctions](#) [Next Autopilot](#)