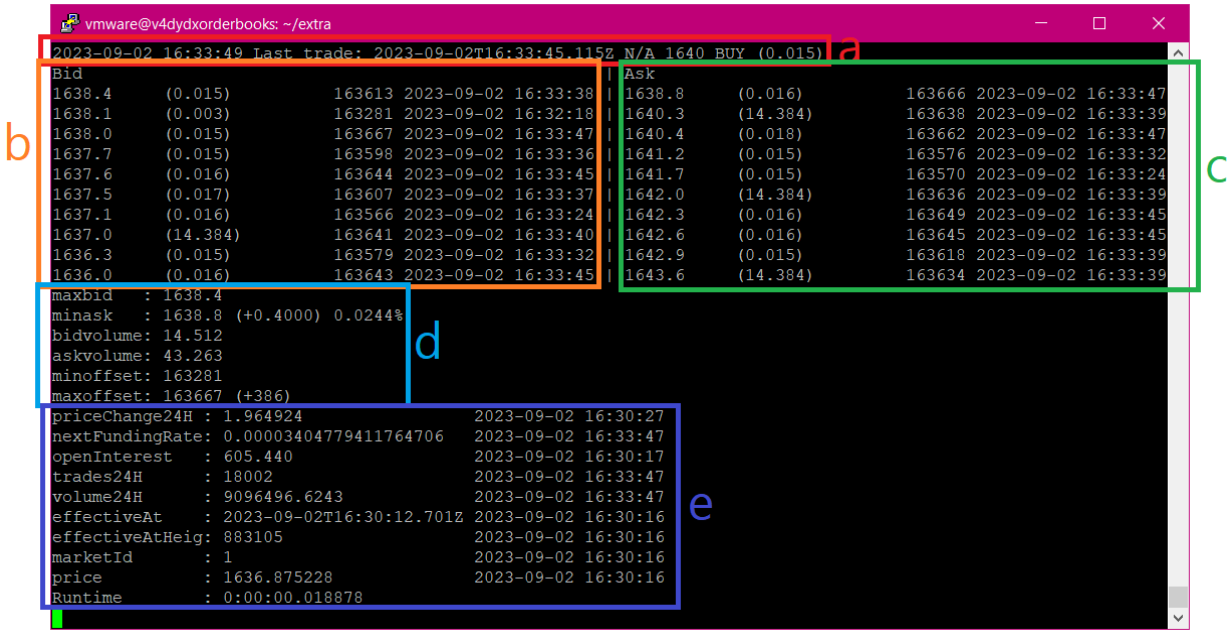
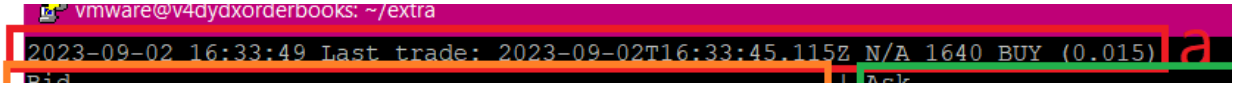
V4 dYdX Orderbook

Author: Lawrence Chiu (lawrence@dydx.exchange)
Twitter: https://twitter.com/LawrenceChiu14
Date: 9/1/2023 (Updated 9/2/2023)
Tested on: Ubuntu Server 20.04.6 LTS

Let's start off by looking at the end result first. This is the order book for ETH-USD. There are 5 areas to talk about, labeled 'a' to 'e' inclusive.

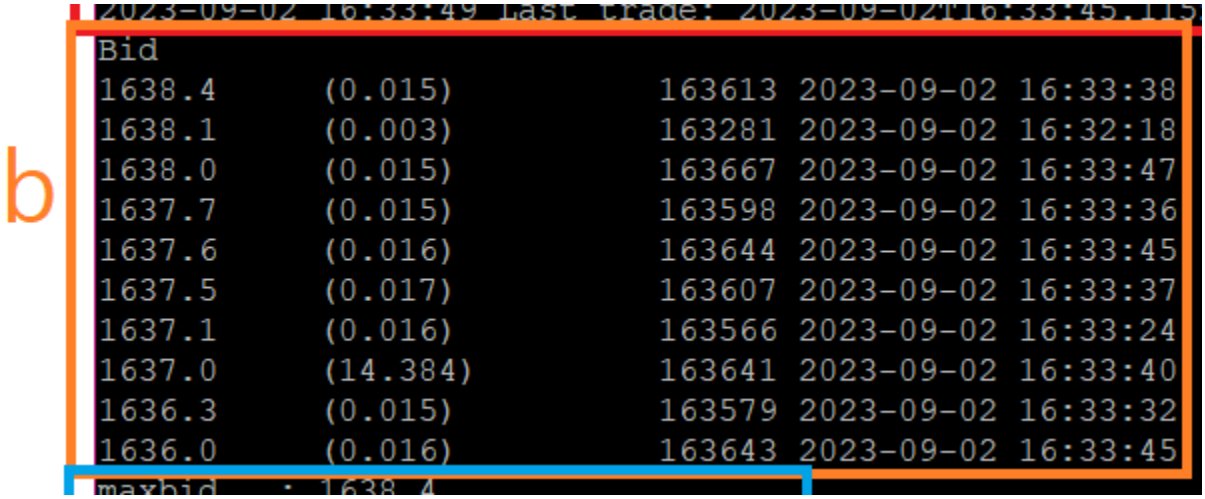


a. This is the last trade from the v4_trades websocket channel.



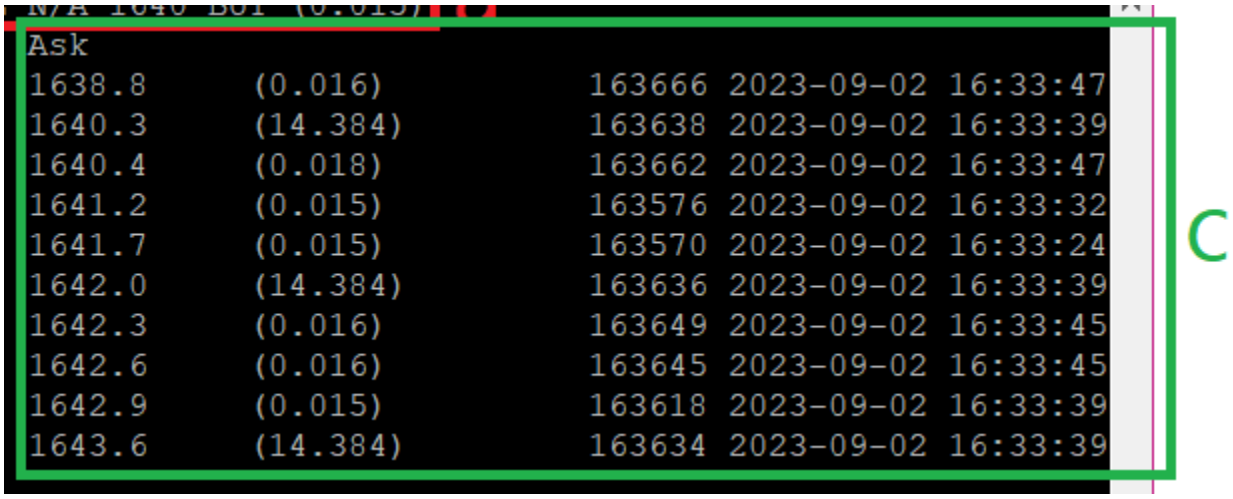
From left to right, we got: 1) local server date and time, then 2) the trade's date and time, 3) either 'N/A' here or the *tradecreatedatheight* value if it is a long-term trade, 4) the price, 5) side (BUY or SELL), and 6) size.

b. This is the list of bids from high price to low price.

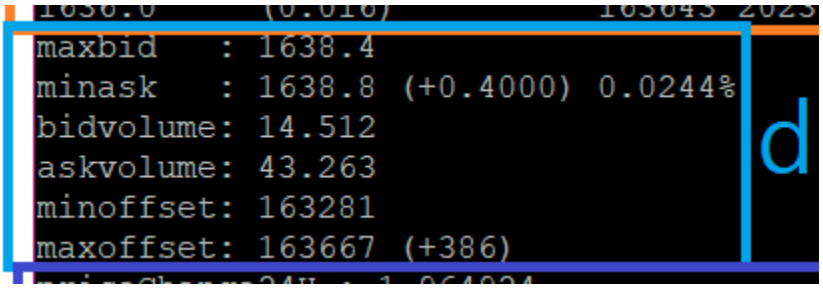


From left to right, we got: 1) price, 2) size, 3) the 'offset' (we'll talk about this below), and 4) the local server date and time (when the price was received on websocket).

c. Same thing as above except these are asks.



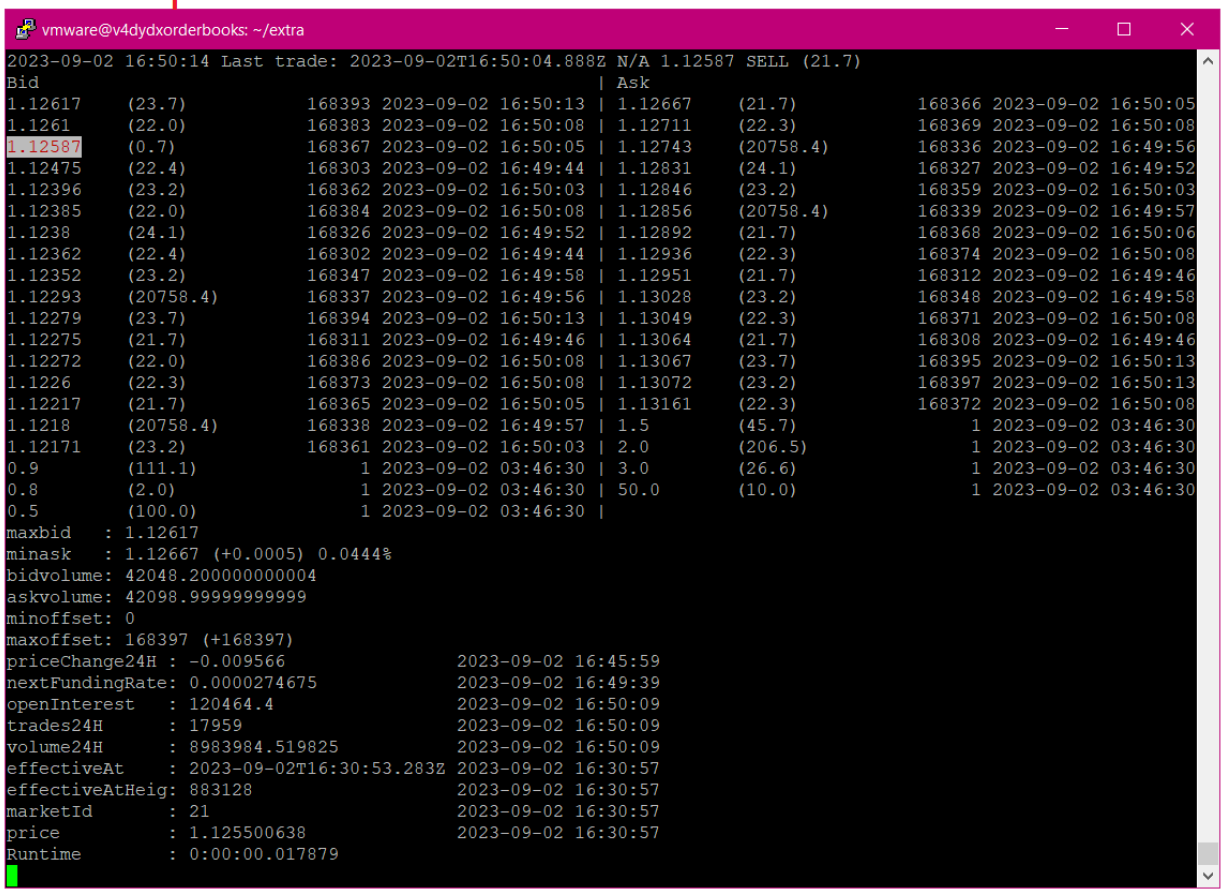
d. This shows some information (some calculated) from the bids/asks above.



In particular, we see: 1) the highest bid, 2) the lowest ask along with a delta both in amount and percent, 3) the total bid volume, 4) the total ask volume, 5) the minimum offset, and 6) the maximum offset along with a delta. Note that these numbers will change if you choose a different depth level. In this example, depth = 10, so we see 10 bids and 10 asks. If you specify a depth that is larger than what's available, it'll show the entire side of the book. For example,

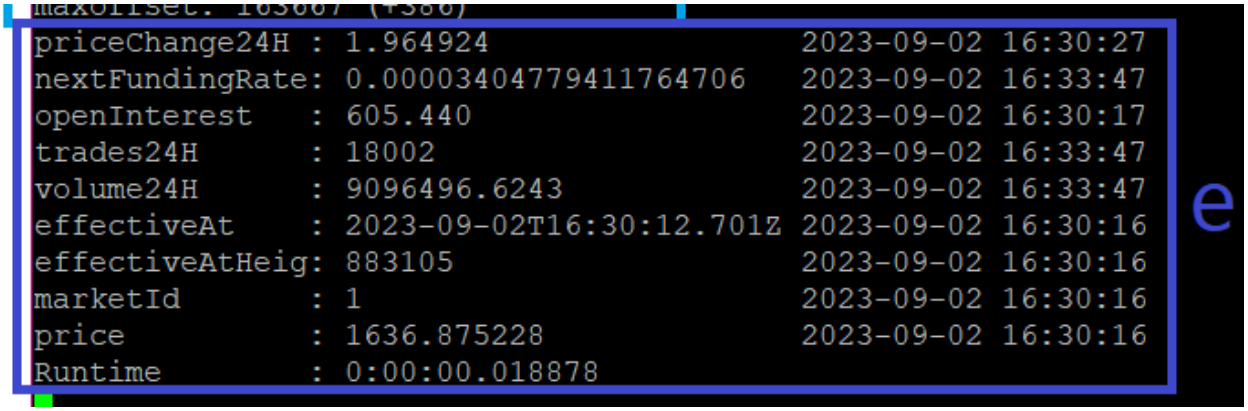
below we see the orderbook for NEAR-USD with depth = 30. The order book only has 29 asks, so it has one fewer level than on the bid side.

Example of an order book with fewer depth on one side vs the other



Let’s continue with our ETH-USD example.

e. This shows market parameters from the v4_markets websocket.



Don't be concerned if you don't see all of these parameters early on. The missing ones simply haven't appeared on the websocket yet, but eventually you'll see all of them. Finally, the last time (Runtime) tells you how long it took to render this data, in this case just 18ms.

Notes

1. Please read the V3 orderbook.pdf first at https://github.com/chiwalfrm/dydxexamples
2. The following are the V4 files with their V3 equivalents. They work exactly the same except on V4: "dydx-testnet-2" (Testnet Chain ID).

V4 program	**V3 program**
v4dydxob.py	dydxob.py
v4dydxtrades.py	dydxtrades.py
v4dydxv4markets.py	dydxv3markets.py
v4dydxob2.py	dydxob2.py

3. In addition, **v4dydxmarkets.py** can be used to list every market and their status (equivalent to **dydxmarkets.py** for V3). There are currently 33 total.

```
vmware@v4dydxorderbooks:~/extra$ python3 v4dydxmarkets.py
BTC-USD ACTIVE
ETH-USD ACTIVE
LINK-USD ACTIVE
MATIC-USD ACTIVE
CRV-USD ACTIVE
SOL-USD ACTIVE
ADA-USD ACTIVE
AVAX-USD ACTIVE
FIL-USD ACTIVE
AAVE-USD ACTIVE
LTC-USD ACTIVE
DOGE-USD ACTIVE
ICP-USD ACTIVE
ATOM-USD ACTIVE
DOT-USD ACTIVE
XTZ-USD ACTIVE
UNI-USD ACTIVE
BCH-USD ACTIVE
EOS-USD ACTIVE
TRX-USD ACTIVE
ALGO-USD ACTIVE
NEAR-USD ACTIVE
SNX-USD ACTIVE
MKR-USD ACTIVE
SUSHI-USD ACTIVE
XLM-USD ACTIVE
XMR-USD ACTIVE
ETC-USD ACTIVE
1INCH-USD ACTIVE
COMP-USD ACTIVE
ZEC-USD ACTIVE
ZRX-USD ACTIVE
YFI-USD ACTIVE
vmware@v4dydxorderbooks:~/extra$
```

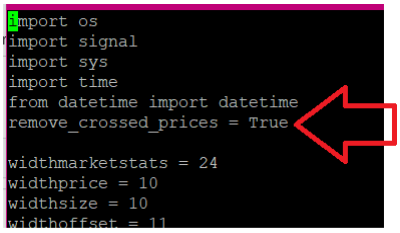
4. The order book display a price in color if it matches the last trade. You can turn this off by specifying ‘noansi’ as the last parameter in the command line.

```
vmware@v4dydxorderbooks: ~/extra
2023-09-02 04:42:21 Last trade: 2023-09-02T04:41:54.903Z N/A 1633.
Bid                                                      | Ask
1633.4     (13.794)          11514 2023-09-02 04:40:36 | 1633.7
1633.3     (0.015)           11847 2023-09-02 04:42:15 | 1633.9
```

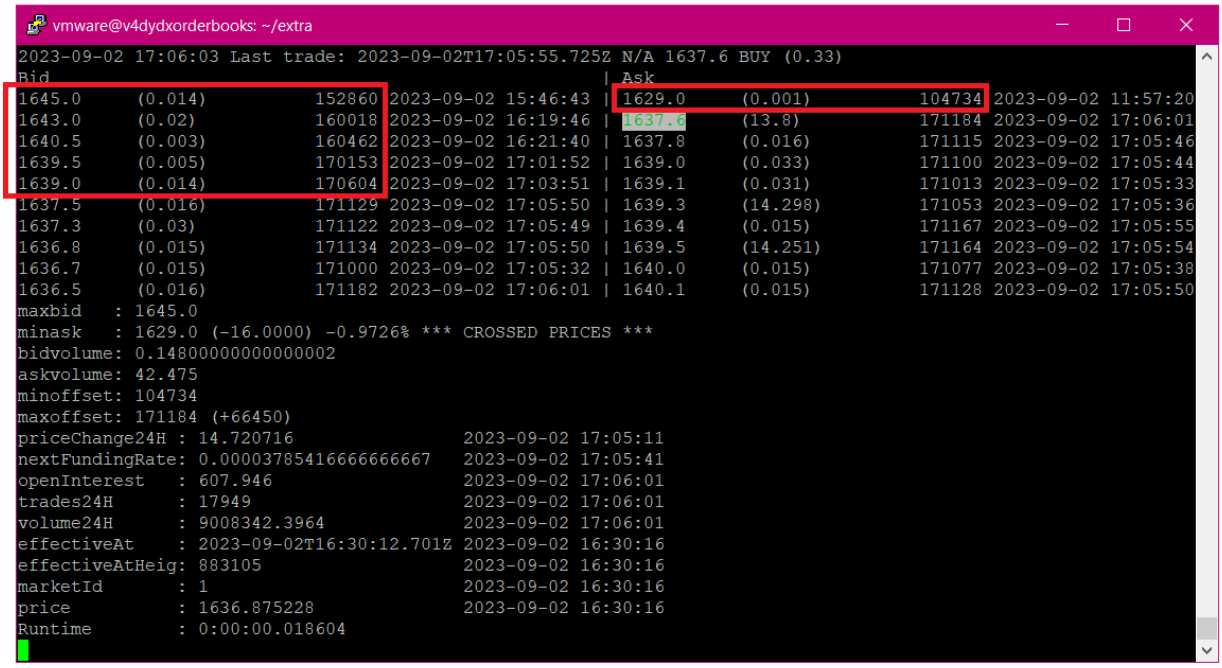
In this example, the highlighted ask (1633.7) is the same price as the last trade.

About ‘offset’

1. Because V4 does not have a centralized order book, there is no longer a global ‘offset’ parameter (like we had in V3) for order book prices. At any given time, the ‘correct’ order book is sitting in the mempool of the current block proposer. The block proposer changes every block, so that means a new canonical mempool and therefore a new canonical order book. Therefore, from time to time, you may see crossed prices from the indexer’s websocket. The way we deal with that in this implementation, is we assign every price, our own local offset (using the websocket’s *message_id*) and we assume that later prices have higher precedence. If you do not want to remove crossed prices, you can change **v4dydxob2.py**’s *remove_crossed_prices* to False



2. Let's take a look at an example of an order book with crossed prices (by setting the parameter above to False).



Here you can see the multiple bids (from 1639 to 1645) crossing, and one asks also. But you can also see these bids and asks have lower offsets (below 171k) than the rest. By default, the program logic (with *remove_crossed_prices* to True) assumes the higher offset takes precedence, and removes those prices.